

BPlusTree

Generated by Doxygen 1.13.2



<b>1 Class Index</b>	<b>1</b>
1.1 Class List	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Class Documentation</b>	<b>5</b>
3.1 Block Class Reference	5
3.1.1 Constructor & Destructor Documentation	6
3.1.1.1 Block()	6
3.1.2 Member Function Documentation	6
3.1.2.1 deserialize()	6
3.1.2.2 dump()	7
3.1.2.3 serialize()	7
3.1.3 Member Data Documentation	7
3.1.3.1 blockNumber	7
3.1.3.2 blockType	8
3.1.3.3 nextBlock	8
3.1.3.4 records	8
3.2 BlockBuffer Class Reference	8
3.2.1 Constructor & Destructor Documentation	9
3.2.1.1 BlockBuffer() [1/2]	9
3.2.1.2 BlockBuffer() [2/2]	9
3.2.2 Member Function Documentation	9
3.2.2.1 getFilename()	9
3.2.2.2 readBlocks()	9
3.2.2.3 setFilename()	10
3.2.2.4 writeBlocks()	10
3.2.3 Member Data Documentation	12
3.2.3.1 filename	12
3.3 BPlusTree Class Reference	12
3.3.1 Detailed Description	13
3.3.2 Constructor & Destructor Documentation	13
3.3.2.1 BPlusTree()	13
3.3.3 Member Function Documentation	14
3.3.3.1 buildTree()	14
3.3.3.2 createIndexLevel()	15
3.3.3.3 deleteRecord()	16
3.3.3.4 dumpTree()	16
3.3.3.5 insert()	16
3.3.3.6 mergeBlocks()	17
3.3.3.7 search()	17
3.3.3.8 splitBlock()	18

3.3.4 Member Data Documentation	19
3.3.4.1 buffer	19
3.3.4.2 filename	19
3.3.4.3 recordsPerBlock	19
3.3.4.4 rootRBN	19
3.3.4.5 totalBlocks	19
3.4 Buffer Class Reference	20
3.4.1 Member Function Documentation	20
3.4.1.1 getBuffer()	20
3.4.1.2 pack()	20
3.4.1.3 readHeader()	21
3.4.1.4 unpack()	21
3.4.1.5 writeHeader()	21
3.4.2 Member Data Documentation	22
3.4.2.1 buffer	22
3.5 Record Class Reference	22
3.5.1 Constructor & Destructor Documentation	22
3.5.1.1 Record()	22
3.5.2 Member Function Documentation	23
3.5.2.1 deserialize()	23
3.5.2.2 serialize()	23
3.5.3 Member Data Documentation	23
3.5.3.1 field1	23
3.5.3.2 field2	24
3.5.3.3 field3	24
3.5.3.4 index	24
<b>4 File Documentation</b>	<b>25</b>
4.1 Block.h File Reference	25
4.1.1 Enumeration Type Documentation	26
4.1.1.1 BlockType	26
4.2 Block.h	27
4.3 BlockBuffer.cpp File Reference	28
4.4 BlockBuffer.h File Reference	28
4.5 BlockBuffer.h	29
4.6 BPlusTree.cpp File Reference	30
4.7 BPlusTree.h File Reference	30
4.8 BPlusTree.h	31
4.9 Buffer.cpp File Reference	32
4.10 Buffer.h File Reference	33
4.11 Buffer.h	34
4.12 main.cpp File Reference	34

---

4.12.1 Function Documentation . . . . .	35
4.12.1.1 createBlocks() . . . . .	35
4.12.1.2 dumpLogical() . . . . .	35
4.12.1.3 dumpPhysical() . . . . .	36
4.12.1.4 main() . . . . .	36
4.12.1.5 readCSV() . . . . .	37
4.13 Record.h File Reference . . . . .	38
4.14 Record.h . . . . .	39
<b>Index</b>	<b>41</b>



# Chapter 1

## Class Index

### 1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">Block</a>	5
<a href="#">BlockBuffer</a>	8
<a href="#">BPlusTree</a>	
Represents a B+ Tree implementation	12
<a href="#">Buffer</a>	20
<a href="#">Record</a>	22





## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">Block.h</a>	25
<a href="#">BlockBuffer.cpp</a>	28
<a href="#">BlockBuffer.h</a>	28
<a href="#">BPlusTree.cpp</a>	30
<a href="#">BPlusTree.h</a>	30
<a href="#">Buffer.cpp</a>	32
<a href="#">Buffer.h</a>	33
<a href="#">main.cpp</a>	34
<a href="#">Record.h</a>	38



## Chapter 3

# Class Documentation

### 3.1 Block Class Reference

```
#include <Block.h>
```

#### Public Member Functions

- [Block](#) ()
- std::string [serialize](#) () const  
*Serializes the block to a string.*
- void [dump](#) () const  
*Dumps the block content to standard output.*

#### Static Public Member Functions

- static [Block deserialize](#) (const std::string &data)  
*Deserializes a block from a string.*

#### Public Attributes

- int [blockNumber](#)  
*Sequential number of the block.*
- int [nextBlock](#)  
*Logical pointer to the next block (-1 if none).*
- [BlockType](#) [blockType](#)  
*Type of the block (LEAF or INDEX).*
- std::vector< [Record](#) > [records](#)  
*List of records in this block.*

### 3.1.1 Constructor & Destructor Documentation

#### 3.1.1.1 Block()

```
Block::Block () [inline]
```

Here is the caller graph for this function:



### 3.1.2 Member Function Documentation

#### 3.1.2.1 deserialize()

```
static Block Block::deserialize (
    const std::string & data) [inline], [static]
```

Deserializes a block from a string.

Expects the first line to be the block header. Each subsequent line is a packed record.

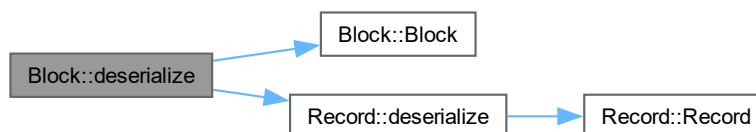
##### Parameters

<i>data</i>	The serialized block string.
-------------	------------------------------

##### Returns

A [Block](#) object.

Here is the call graph for this function:



Here is the caller graph for this function:

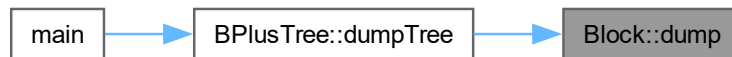


### 3.1.2.2 dump()

```
void Block::dump () const [inline]
```

Dumps the block content to standard output.

Here is the caller graph for this function:



### 3.1.2.3 serialize()

```
std::string Block::serialize () const [inline]
```

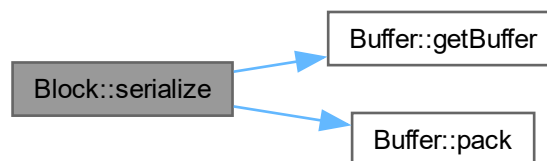
Serializes the block to a string.

First writes a header line: blockNumber,blockType,recordCount,nextBlock Then, for each record, packs the record using [Buffer](#) and writes the result.

#### Returns

The serialized block string.

Here is the call graph for this function:



## 3.1.3 Member Data Documentation

### 3.1.3.1 blockNumber

```
int Block::blockNumber
```

Sequential number of the block.

### 3.1.3.2 blockType

```
BlockType Block::blockType
```

Type of the block (LEAF or INDEX).

### 3.1.3.3 nextBlock

```
int Block::nextBlock
```

Logical pointer to the next block (-1 if none).

### 3.1.3.4 records

```
std::vector<Record> Block::records
```

List of records in this block.

The documentation for this class was generated from the following file:

- [Block.h](#)

## 3.2 BlockBuffer Class Reference

```
#include <BlockBuffer.h>
```

### Public Member Functions

- [BlockBuffer](#) ()=default  
*Default constructor.*
- [BlockBuffer](#) (const std::string &file)  
*Constructor that accepts a filename for file operations.*
- bool [writeBlocks](#) (const std::string &filename, const std::vector< [Block](#) > &blocks)  
*Writes a blocked sequence set file.*
- bool [readBlocks](#) (const std::string &filename, std::vector< [Block](#) > &blocks)  
*Reads a blocked sequence set file.*
- std::string [getFilename](#) () const  
*Accessor for the filename used in [BlockBuffer](#).*
- void [setFilename](#) (const std::string &file)  
*Mutator to set the filename for the [BlockBuffer](#).*

### Private Attributes

- std::string [filename](#)

## 3.2.1 Constructor & Destructor Documentation

### 3.2.1.1 BlockBuffer() [1/2]

```
BlockBuffer::BlockBuffer () [default]
```

Default constructor.

### 3.2.1.2 BlockBuffer() [2/2]

```
BlockBuffer::BlockBuffer (  
    const std::string & file) [inline], [explicit]
```

Constructor that accepts a filename for file operations.

#### Parameters

<i>file</i>	The file name to initialize the <a href="#">BlockBuffer</a> with.
-------------	---

## 3.2.2 Member Function Documentation

### 3.2.2.1 getFilename()

```
std::string BlockBuffer::getFilename () const [inline]
```

Accessor for the filename used in [BlockBuffer](#).

#### Returns

The file name associated with this [BlockBuffer](#).

### 3.2.2.2 readBlocks()

```
bool BlockBuffer::readBlocks (  
    const std::string & filename,  
    std::vector< Block > & blocks)
```

Reads a blocked sequence set file.

#### Parameters

<i>filename</i>	The input file name.
<i>blocks</i>	A vector to receive the blocks.

#### Returns

true on success.

Reads blocks from a file, deserializing each block after unpacking. Expects the file to have a header specifying the format and the number of blocks. Each block is read using its size indicator and reconstructed.

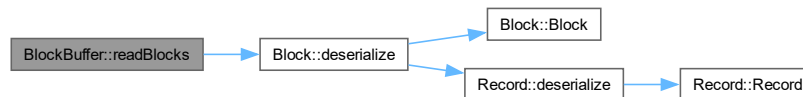
**Parameters**

<i>filename</i>	The name of the input file to read the blocks from.
<i>blocks</i>	A vector to receive the deserialized blocks.

**Returns**

True if the operation succeeds, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.2.2.3 setFilename()**

```
void BlockBuffer::setFilename (
    const std::string & file) [inline]
```

Mutator to set the filename for the [BlockBuffer](#).

**Parameters**

<i>file</i>	The new file name to set.
-------------	---------------------------

**3.2.2.4 writeBlocks()**

```
bool BlockBuffer::writeBlocks (
    const std::string & filename,
    const std::vector< Block > & blocks)
```

Writes a blocked sequence set file.

The file consists of:

- A file header (packed using [Buffer](#))
- A line with the number of blocks
- For each block: a length indicator (the size of the packed block) and the packed block data.



## Parameters

<i>filename</i>	The output file name.
<i>blocks</i>	A vector of blocks to write.

## Returns

true on success.

Writes blocks to a file, including a header that specifies the format and the number of blocks in the file. Each block is serialized, packed using [Buffer](#), and written alongside its size indicator.

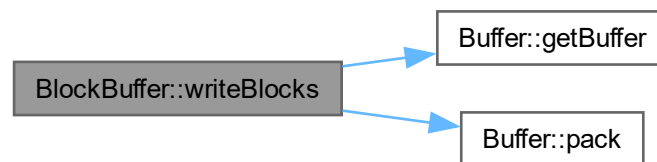
## Parameters

<i>filename</i>	The name of the output file to write the blocks to.
<i>blocks</i>	A vector of <a href="#">Block</a> objects to write to the file.

## Returns

True if the operation succeeds, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.2.3 Member Data Documentation

#### 3.2.3.1 filename

```
std::string BlockBuffer::filename [private]
```

The documentation for this class was generated from the following files:

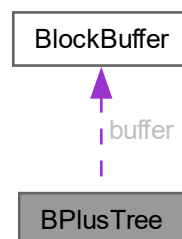
- [BlockBuffer.h](#)
- [BlockBuffer.cpp](#)

## 3.3 BPlusTree Class Reference

Represents a B+ Tree implementation.

```
#include <BPlusTree.h>
```

Collaboration diagram for BPlusTree:



### Public Member Functions

- [BPlusTree](#) (const std::string &filename, int recordsPerBlock)  
*Constructor for the [BPlusTree](#) class.*
- void [buildTree](#) (const std::vector< [Block](#) > &blocks)  
*Constructs the B+ Tree hierarchy from a set of leaf blocks.*
- void [insert](#) (const [Record](#) &record)  
*Dynamically inserts a new record into the B+ Tree.*
- void [deleteRecord](#) (int key)  
*Dynamically deletes a record from the B+ Tree.*
- [Record search](#) (int key)  
*Searches for a record by key in the B+ Tree.*
- void [dumpTree](#) ()  
*Dumps the structure of the B+ Tree for debugging purposes.*

### Private Member Functions

- int [createIndexLevel](#) (const std::vector< int > &keys, const std::vector< int > &childRBNs)  
*Recursively creates index levels for the B+ Tree.*
- void [splitBlock](#) ([Block](#) &block, int parentRBN)  
*Splits a block on insertion.*
- void [mergeBlocks](#) ([Block](#) &leftBlock, [Block](#) &rightBlock, int parentRBN)  
*Merges two blocks on deletion.*

### Private Attributes

- int [rootRBN](#)  
*Relative [Block](#) Number of the root block.*
- int [recordsPerBlock](#)  
*Maximum number of records allowed per block.*
- int [totalBlocks](#)  
*Total number of blocks currently in the tree.*
- std::string [filename](#)  
*Name of the file where the B+ Tree is stored.*
- [BlockBuffer](#) [buffer](#)  
*Manages file I/O operations for blocks.*

## 3.3.1 Detailed Description

Represents a B+ Tree implementation.

Provides methods to construct, manage, and query a B+ Tree. This class handles file-based storage for scalability, supporting dynamic insertion, deletion, and search operations.

## 3.3.2 Constructor & Destructor Documentation

### 3.3.2.1 BPlusTree()

```
BPlusTree::BPlusTree (
    const std::string & filename,
    int recordsPerBlock)
```

Constructor for the [BPlusTree](#) class.

Initializes the B+ Tree using the specified file and record-per-block settings. Reads existing metadata from the file if available, or starts a new tree if not.

#### Parameters

<i>filename</i>	The name of the file associated with the B+ Tree.
<i>recordsPerBlock</i>	The maximum number of records allowed per block.

Initializes the B+ Tree from a file or starts a new tree if the file does not exist. The constructor attempts to read the file header to retrieve metadata such as `recordsPerBlock`, `totalBlocks`, and `rootRBN`.

## Parameters

<i>filename</i>	The name of the file associated with the B+ Tree.
<i>recordsPerBlock</i>	The maximum number of records allowed per block.

### 3.3.3 Member Function Documentation

#### 3.3.3.1 buildTree()

```
void BPlusTree::buildTree (
    const std::vector< Block > & blocks)
```

Constructs the B+ Tree hierarchy from a set of leaf blocks.

Builds the B+ Tree from a set of leaf blocks.

Writes all leaf blocks to the file and recursively creates index levels until the root node is formed.

## Parameters

<i>blocks</i>	A vector of leaf blocks to form the base of the tree.
---------------	---

Writes all leaf blocks to the file and constructs the hierarchical index levels to organize the tree. The root block number is updated after constructing the index levels.

## Parameters

<i>blocks</i>	A vector of leaf blocks to form the base of the B+ Tree.
---------------	--

Here is the call graph for this function:



Here is the caller graph for this function:



## 3.3.3.2 createIndexLevel()

```
int BPlusTree::createIndexLevel (
    const std::vector< int > & keys,
    const std::vector< int > & childRBNs) [private]
```

Recursively creates index levels for the B+ Tree.

Recursively creates hierarchical index levels for the B+ Tree.

Aggregates keys and child block numbers into parent index blocks, continuing until a single root node is formed.

## Parameters

<i>keys</i>	A vector of keys representing the largest keys from child blocks.
<i>childRBNs</i>	A vector of Relative <a href="#">Block</a> Numbers (RBNs) for the child blocks.

## Returns

The RBN of the root index block created.

Aggregates keys and block numbers from child blocks into parent index blocks. Continues building levels until a single root index block is formed.

## Parameters

<i>keys</i>	A vector of keys representing the largest keys from child blocks.
<i>childRBNs</i>	A vector of Relative <a href="#">Block</a> Numbers (RBNs) for the child blocks.

## Returns

The RBN of the root index block created.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.3.3.3 deleteRecord()

```
void BPlusTree::deleteRecord (  
    int key)
```

Dynamically deletes a record from the B+ Tree.

Removes the record with the specified key from the tree. Handles block underflows by merging blocks and updating the tree structure.

#### Parameters

<i>key</i>	The key of the record to delete.
------------	----------------------------------

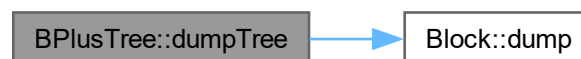
### 3.3.3.4 dumpTree()

```
void BPlusTree::dumpTree ()
```

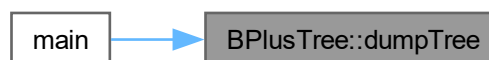
Dumps the structure of the B+ Tree for debugging purposes.

Traverses the tree and prints the contents of each block, starting from the root block.

Traverses the tree and prints the contents of each block, starting from the root block. Displays both leaf and index blocks. Here is the call graph for this function:



Here is the caller graph for this function:



### 3.3.3.5 insert()

```
void BPlusTree::insert (  
    const Record & record)
```

Dynamically inserts a new record into the B+ Tree.

Inserts a new record into the B+ Tree.

Finds the appropriate leaf block for the new record and inserts it. Handles block overflows by splitting blocks and updating the tree structure as needed.

## Parameters

<i>record</i>	The record to be inserted into the tree.
---------------	--

Finds the appropriate leaf block for the new record and inserts it. Handles overflow by splitting blocks as necessary and updating the parent blocks.

## Parameters

<i>record</i>	The record to be inserted into the tree.
---------------	--

Here is the call graph for this function:



## 3.3.3.6 mergeBlocks()

```
void BPlusTree::mergeBlocks (
    Block & leftBlock,
    Block & rightBlock,
    int parentRBN) [private]
```

Merges two blocks on deletion.

When a block underflows, its records are merged with a sibling block. Updates the parent block to reflect the merged structure.

## Parameters

<i>leftBlock</i>	The left sibling block.
<i>rightBlock</i>	The right sibling block.
<i>parentRBN</i>	The RBN of the parent block.

## 3.3.3.7 search()

```
Record BPlusTree::search (
    int key)
```

Searches for a record by key in the B+ Tree.

Traverses the tree to locate the record with the specified key, starting from the root block.

## Parameters

<i>key</i>	The key to search for.
------------	------------------------

## Returns

The record associated with the key, or an empty record if not found.

**3.3.3.8 splitBlock()**

```
void BPlusTree::splitBlock (
    Block & block,
    int parentRBN) [private]
```

Splits a block on insertion.

Splits an overflowing block into two balanced blocks.

When a block overflows, its records are redistributed into two balanced blocks. Handles updates to parent blocks or creates a new root block if needed.

## Parameters

<i>block</i>	The overflowing block to split.
<i>parentRBN</i>	The RBN of the parent block, or -1 if the block has no parent.

When a block overflows, its records are redistributed into a new block, maintaining the balance of the B+ Tree. If there is no parent block, a new root is created to reference the split blocks.

## Parameters

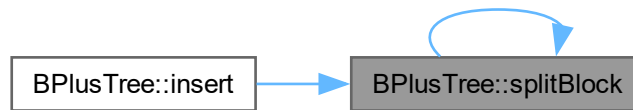
<i>block</i>	The overflowing block that needs to be split.
<i>parentRBN</i>	The Relative <a href="#">Block</a> Number (RBN) of the parent block, or -1 if no parent exists.

Here is the call graph for this function:





Here is the caller graph for this function:



### 3.3.4 Member Data Documentation

#### 3.3.4.1 buffer

```
BlockBuffer BPlusTree::buffer [private]
```

Manages file I/O operations for blocks.

#### 3.3.4.2 filename

```
std::string BPlusTree::filename [private]
```

Name of the file where the B+ Tree is stored.

#### 3.3.4.3 recordsPerBlock

```
int BPlusTree::recordsPerBlock [private]
```

Maximum number of records allowed per block.

#### 3.3.4.4 rootRBN

```
int BPlusTree::rootRBN [private]
```

Relative [Block](#) Number of the root block.

#### 3.3.4.5 totalBlocks

```
int BPlusTree::totalBlocks [private]
```

Total number of blocks currently in the tree.

The documentation for this class was generated from the following files:

- [BPlusTree.h](#)
- [BPlusTree.cpp](#)

## 3.4 Buffer Class Reference

```
#include <Buffer.h>
```

### Public Member Functions

- void [pack](#) (const std::string &data)  
*Packs a string into a length-indicated format.*
- std::string [unpack](#) ()  
*Unpacks the string (ignores the length indicator).*
- void [readHeader](#) (std::ifstream &file)  
*Reads the header record from the input file stream.*
- void [writeHeader](#) (std::ofstream &file)  
*Writes the header record to the output file stream.*
- std::string [getBuffer](#) () const  
*Returns the internal packed string.*

### Private Attributes

- std::string [buffer](#)

### 3.4.1 Member Function Documentation

#### 3.4.1.1 [getBuffer\(\)](#)

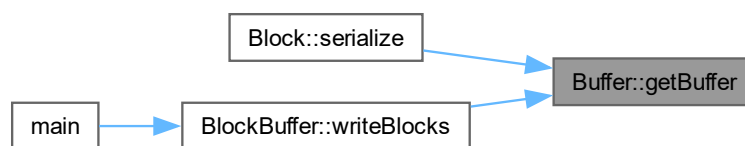
```
std::string Buffer::getBuffer () const [inline]
```

Returns the internal packed string.

#### Returns

The packed string.

Here is the caller graph for this function:



#### 3.4.1.2 [pack\(\)](#)

```
void Buffer::pack (
    const std::string & data)
```

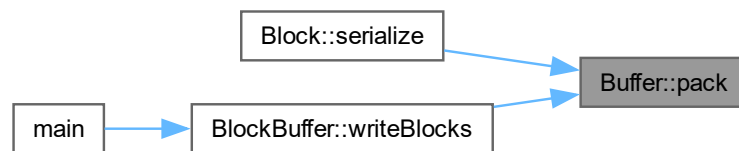
Packs a string into a length-indicated format.

Example: "Hello" becomes "5,Hello"

## Parameters

<i>data</i>	The string to pack.
-------------	---------------------

Here is the caller graph for this function:



## 3.4.1.3 readHeader()

```
void Buffer::readHeader (
    std::ifstream & file)
```

Reads the header record from the input file stream.

## Parameters

<i>file</i>	The input stream.
-------------	-------------------

## 3.4.1.4 unpack()

```
std::string Buffer::unpack ()
```

Unpacks the string (ignores the length indicator).

Unpacks a length-indicated string from the buffer.

## Returns

The original string.

## 3.4.1.5 writeHeader()

```
void Buffer::writeHeader (
    std::ofstream & file)
```

Writes the header record to the output file stream.

## Parameters

<i>file</i>	The output stream.
-------------	--------------------

### 3.4.2 Member Data Documentation

#### 3.4.2.1 buffer

```
std::string Buffer::buffer [private]
```

The documentation for this class was generated from the following files:

- [Buffer.h](#)
- [Buffer.cpp](#)

## 3.5 Record Class Reference

```
#include <Record.h>
```

### Public Member Functions

- [Record](#) ()
- std::string [serialize](#) () const  
*Serializes the record as a CSV string.*

### Static Public Member Functions

- static [Record deserialize](#) (const std::string &data)  
*Deserializes a CSV string into a [Record](#).*

### Public Attributes

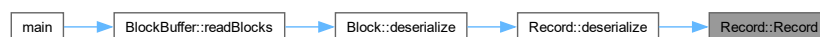
- int [index](#)
- std::string [field1](#)
- std::string [field2](#)
- std::string [field3](#)

### 3.5.1 Constructor & Destructor Documentation

#### 3.5.1.1 Record()

```
Record::Record () [inline]
```

Here is the caller graph for this function:



## 3.5.2 Member Function Documentation

### 3.5.2.1 deserialize()

```
static Record Record::deserialize (
    const std::string & data) [inline], [static]
```

Deserializes a CSV string into a [Record](#).

#### Parameters

<i>data</i>	The CSV string.
-------------	-----------------

#### Returns

A [Record](#) object.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.5.2.2 serialize()

```
std::string Record::serialize () const [inline]
```

Serializes the record as a CSV string.

Format: index,field1,field2,field3

## 3.5.3 Member Data Documentation

### 3.5.3.1 field1

```
std::string Record::field1
```

### 3.5.3.2 field2

```
std::string Record::field2
```

### 3.5.3.3 field3

```
std::string Record::field3
```

### 3.5.3.4 index

```
int Record::index
```

The documentation for this class was generated from the following file:

- [Record.h](#)

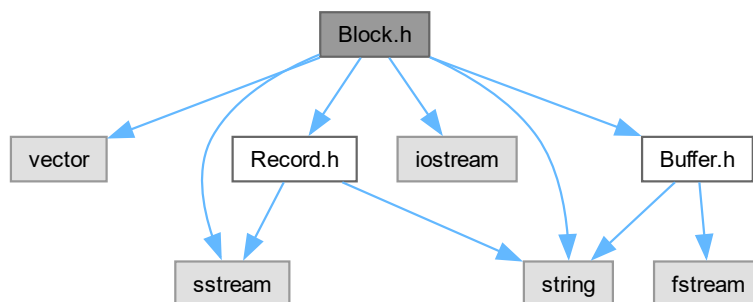
## Chapter 4

# File Documentation

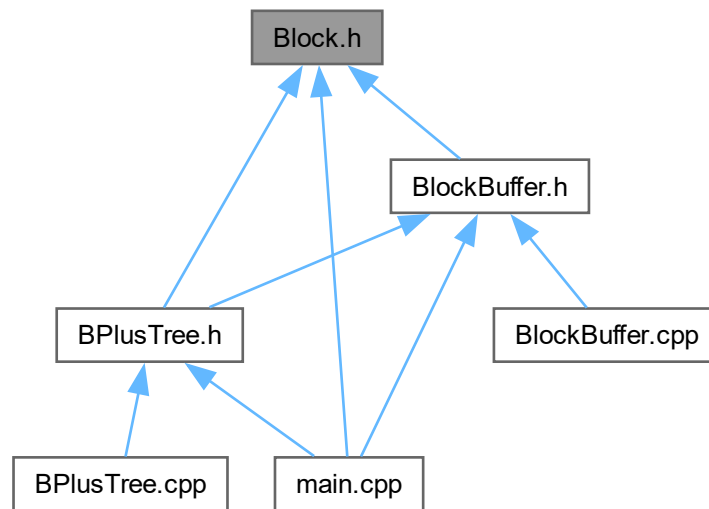
### 4.1 Block.h File Reference

```
#include <vector>
#include <string>
#include <sstream>
#include <iostream>
#include "Record.h"
#include "Buffer.h"
```

Include dependency graph for Block.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [Block](#)

## Enumerations

- enum class [BlockType](#) { [LEAF](#) , [INDEX](#) }

### 4.1.1 Enumeration Type Documentation

#### 4.1.1.1 BlockType

```
enum class BlockType [strong]
```

##### Enumerator

LEAF	
INDEX	



## 4.2 Block.h

[Go to the documentation of this file.](#)

```

00001 #ifndef BLOCK_H
00002 #define BLOCK_H
00003
00004 #include <vector>
00005 #include <string>
00006 #include <sstream>
00007 #include <iostream>
00008 #include "Record.h"
00009 #include "Buffer.h"
00010
00011 // Enum for BlockType
00012 enum class BlockType
00013 {
00014     LEAF, // Leaf blocks contain records
00015     INDEX // Index blocks contain key-pointer pairs
00016 };
00017
00018 class Block
00019 {
00020 public:
00021     int blockNumber;
00022     int nextBlock;
00023     BlockType blockType;
00024     std::vector<Record> records;
00025
00026     // Default Constructor
00027     Block() : blockNumber(0), nextBlock(-1), blockType(BlockType::LEAF) {}
00028
00029     std::string serialize() const
00030     {
00031         std::stringstream ss;
00032         // Write block header
00033         ss << blockNumber << "," << static_cast<int>(blockType) << "," << records.size() << "," << nextBlock
00034         << "\n";
00035
00036         // Write each record (packed with Buffer)
00037         for (const auto &rec : records)
00038         {
00039             Buffer buf;
00040             std::string recStr = rec.serialize();
00041             buf.pack(recStr);
00042             ss << buf.getBuffer() << "\n";
00043         }
00044         return ss.str();
00045     }
00046
00047     static Block deserialize(const std::string &data)
00048     {
00049         Block blk;
00050         std::stringstream ss(data);
00051         std::string line;
00052         // Get header line
00053         if (getline(ss, line))
00054         {
00055             std::stringstream headerStream(line);
00056             std::string token;
00057             getline(headerStream, token, ',');
00058             blk.blockNumber = std::stoi(token);
00059             getline(headerStream, token, ',');
00060             blk.blockType = static_cast<BlockType>(std::stoi(token)); // Convert int to BlockType
00061             getline(headerStream, token, ','); // Record count (not used here)
00062             blk.nextBlock = std::stoi(token);
00063         }
00064         // Read each packed record
00065         while (getline(ss, line))
00066         {
00067             if (line.empty())
00068                 continue;
00069             // Unpack the record manually
00070             size_t commaPos = line.find(',');
00071             if (commaPos == std::string::npos)
00072                 continue;
00073             int len = std::stoi(line.substr(0, commaPos));
00074             std::string recData = line.substr(commaPos + 1, len);
00075             Record r = Record::deserialize(recData);
00076             blk.records.push_back(r);
00077         }
00078         return blk;
00079     }
00080
00081     void dump() const
00082     {

```

```

00102         std::cout << "Block Number: " << blockNumber << ", Next Block: " << nextBlock
00103             << ", Type: " << (blockType == BlockType::LEAF ? "LEAF" : "INDEX") << std::endl;
00104         std::cout << "Records:" << std::endl;
00105         for (const auto &r : records)
00106         {
00107             std::cout << r.index << " | " << r.field1 << " | " << r.field2 << " | " << r.field3 << std::endl;
00108         }
00109     }
00110 };
00111
00112 #endif

```

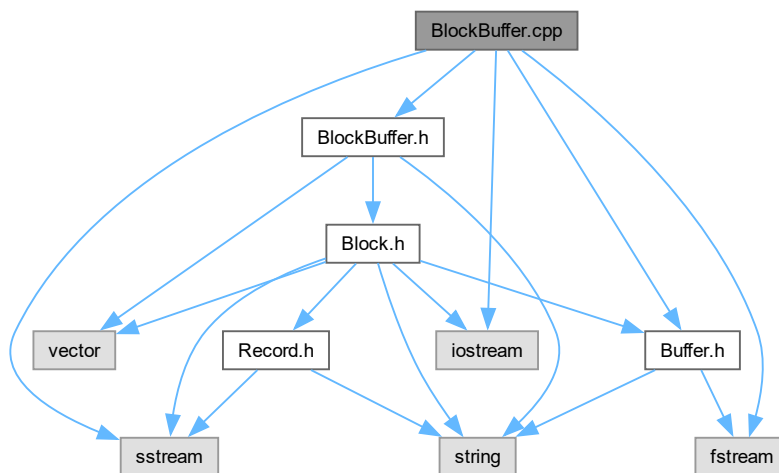
### 4.3 BlockBuffer.cpp File Reference

```

#include "BlockBuffer.h"
#include "Buffer.h"
#include <fstream>
#include <sstream>
#include <iostream>

```

Include dependency graph for BlockBuffer.cpp:



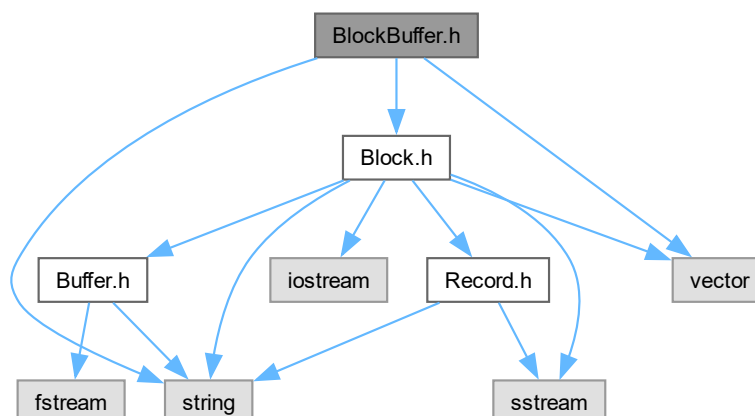
### 4.4 BlockBuffer.h File Reference

```

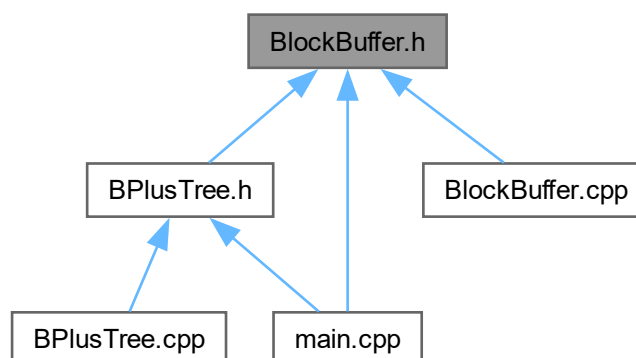
#include <string>
#include <vector>
#include "Block.h"

```

Include dependency graph for BlockBuffer.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [BlockBuffer](#)

## 4.5 BlockBuffer.h

[Go to the documentation of this file.](#)

```

00001 #ifndef BLOCKBUFFER_H
00002 #define BLOCKBUFFER_H
00003

```

```

00004 #include <string>
00005 #include <vector>
00006 #include "Block.h"
00007
00008 class BlockBuffer
00009 {
00010 private:
00011     std::string filename; // Store the file name for file operations
00012
00013 public:
00014     BlockBuffer() = default;
00015
00016     explicit BlockBuffer(const std::string &file) : filename(file) {}
00017
00018     bool writeBlocks(const std::string &filename, const std::vector<Block> &blocks);
00019
00020     bool readBlocks(const std::string &filename, std::vector<Block> &blocks);
00021
00022     std::string getFilename() const { return filename; }
00023
00024     void setFilename(const std::string &file) { filename = file; }
00025 };
00026 #endif

```

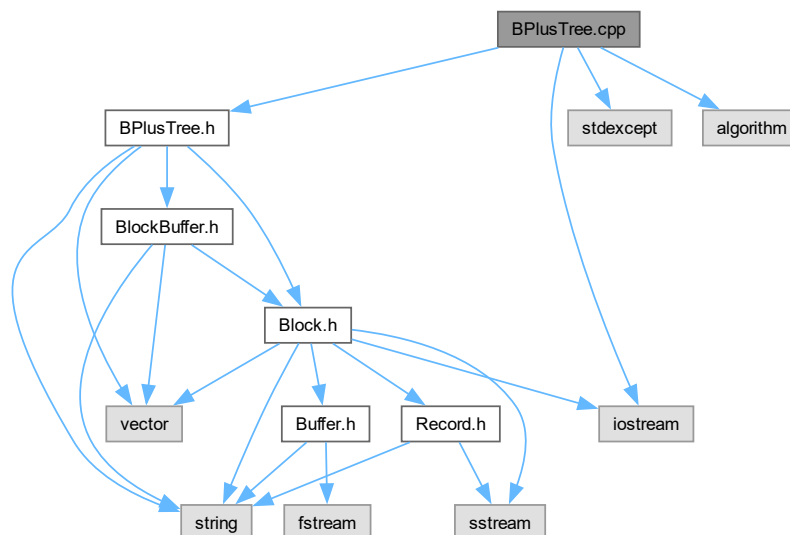
## 4.6 BPlusTree.cpp File Reference

```

#include "BPlusTree.h"
#include <iostream>
#include <stdexcept>
#include <algorithm>

```

Include dependency graph for BPlusTree.cpp:



## 4.7 BPlusTree.h File Reference

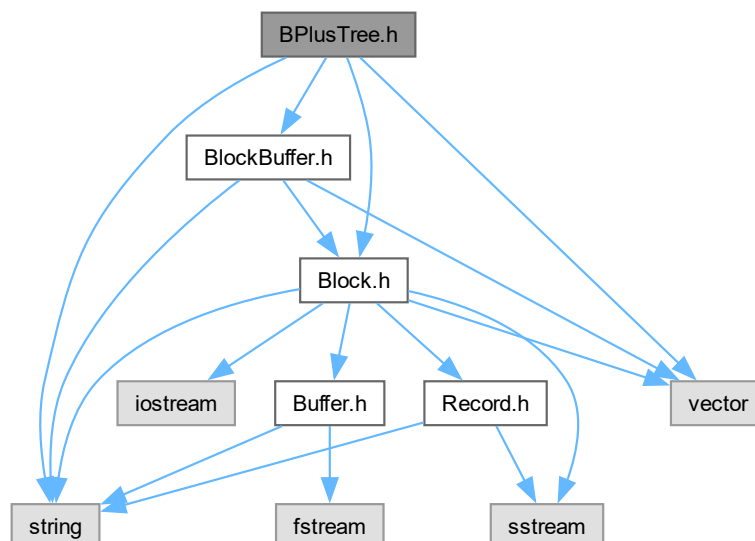
```

#include <string>
#include <vector>
#include "Block.h"

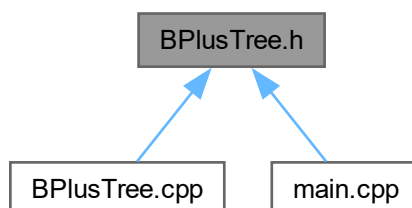
```

```
#include "BlockBuffer.h"
```

Include dependency graph for BPlusTree.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [BPlusTree](#)

*Represents a B+ Tree implementation.*

## 4.8 BPlusTree.h

[Go to the documentation of this file.](#)

```
00001 #ifndef BPLUSTREE_H
```

```

00002 #define BPLUSTREE_H
00003
00004 #include <string>
00005 #include <vector>
00006 #include "Block.h"
00007 #include "BlockBuffer.h"
00008
00016 class BPlusTree
00017 {
00018 public:
00028     BPlusTree(const std::string &filename, int recordsPerBlock);
00029
00038     void buildTree(const std::vector<Block> &blocks);
00039
00048     void insert(const Record &record);
00049
00058     void deleteRecord(int key);
00059
00069     Record search(int key);
00070
00077     void dumpTree();
00078
00079 private:
00080     // Metadata
00081     int rootRBN;
00082     int recordsPerBlock;
00083     int totalBlocks;
00084     std::string filename;
00085
00086     // File handler
00087     BlockBuffer buffer;
00088
00099     int createIndexLevel(const std::vector<int> &keys, const std::vector<int> &childRBNs);
00100
00110     void splitBlock(Block &block, int parentRBN);
00111
00122     void mergeBlocks(Block &leftBlock, Block &rightBlock, int parentRBN);
00123 };
00124
00125 #endif

```

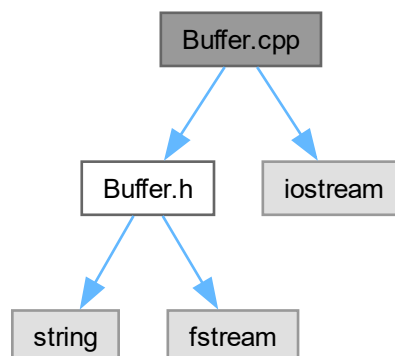
## 4.9 Buffer.cpp File Reference

```

#include "Buffer.h"
#include <iostream>

```

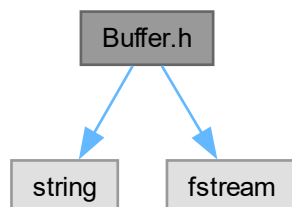
Include dependency graph for Buffer.cpp:



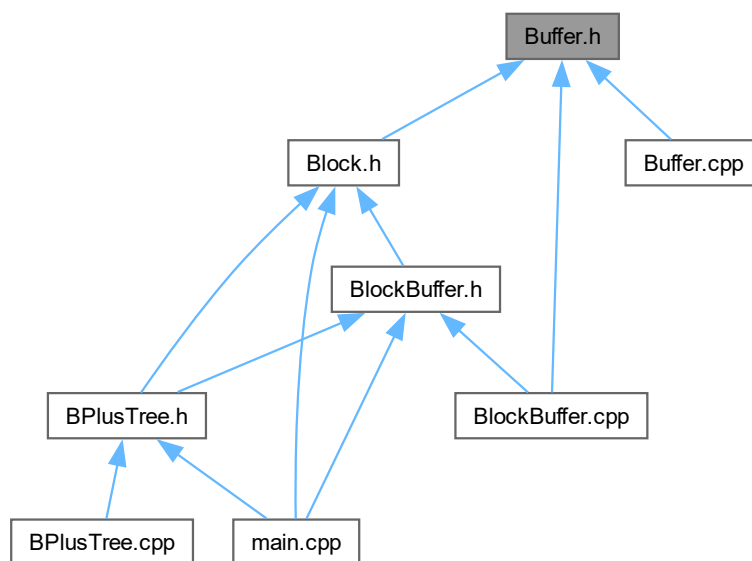
## 4.10 Buffer.h File Reference

```
#include <string>
#include <fstream>
```

Include dependency graph for Buffer.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [Buffer](#)

## 4.11 Buffer.h

[Go to the documentation of this file.](#)

```

00001 #ifndef BUFFER_H
00002 #define BUFFER_H
00003
00004 #include <string>
00005 #include <fstream>
00006
00007 class Buffer {
00008 public:
00016     void pack(const std::string& data);
00017
00023     std::string unpack();
00024
00030     void readHeader(std::ifstream& file);
00031
00037     void writeHeader(std::ofstream& file);
00038
00044     std::string getBuffer() const { return buffer; }
00045
00046 private:
00047     std::string buffer;
00048 };
00049
00050 #endif

```

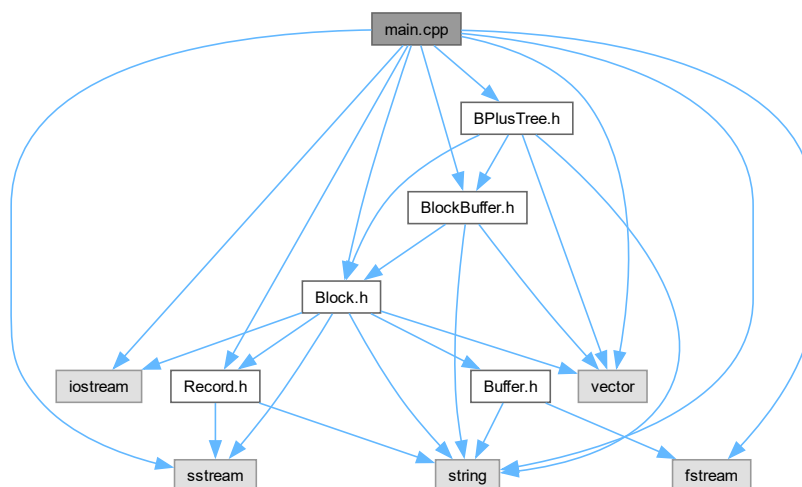
## 4.12 main.cpp File Reference

```

#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <string>
#include "BlockBuffer.h"
#include "Block.h"
#include "Record.h"
#include "BPlusTree.h"

```

Include dependency graph for main.cpp:





## Functions

- vector< string > [readCSV](#) (const string &filename)  
*Reads a CSV file (with a header) and returns a vector of CSV record strings.*
- vector< [Block](#) > [createBlocks](#) (const vector< string > &records, int recordsPerBlock)  
*Creates blocks from CSV record strings.*
- void [dumpPhysical](#) (const vector< [Block](#) > &blocks)  
*Dump blocks in physical order (as stored in file).*
- void [dumpLogical](#) (const vector< [Block](#) > &blocks)  
*Dump blocks in logical order (following nextBlock pointer).*
- int [main](#) (int argc, char \*argv[])  
*Main entry point for the program.*

## 4.12.1 Function Documentation

### 4.12.1.1 createBlocks()

```
vector< Block > createBlocks (
    const vector< string > & records,
    int recordsPerBlock)
```

Creates blocks from CSV record strings.

Splits the CSV records into fixed-sized blocks, with each block containing a specified maximum number of records. Each block is sequentially numbered.

#### Parameters

<i>records</i>	A vector of CSV record strings to split into blocks.
<i>recordsPerBlock</i>	The maximum number of records allowed in each block.

#### Returns

A vector of [Block](#) objects containing the CSV records.

Here is the caller graph for this function:



### 4.12.1.2 dumpLogical()

```
void dumpLogical (
    const vector< Block > & blocks)
```

Dump blocks in logical order (following nextBlock pointer).

Traverses the blocks starting from the first block and follows the `nextBlock` pointers to dump the logical structure of the blocks.

**Parameters**

<i>blocks</i>	A vector of <a href="#">Block</a> objects to dump.
---------------	--

Here is the caller graph for this function:

**4.12.1.3 dumpPhysical()**

```
void dumpPhysical (  
    const vector< Block > & blocks)
```

Dump blocks in physical order (as stored in file).

Iterates through the provided blocks and prints their contents sequentially.

**Parameters**

<i>blocks</i>	A vector of <a href="#">Block</a> objects to dump.
---------------	--

Here is the caller graph for this function:

**4.12.1.4 main()**

```
int main (  
    int argc,  
    char * argv[])
```

Main entry point for the program.

Parses command-line flags, reads records from a CSV file, and generates a blocked sequence set and a B+ Tree. Also supports dumping the physical and logical structure of blocks.

Command-line flags:

- `--dumpPhysical`: Dumps the physical structure of the blocks.
- `--dumpLogical`: Dumps the logical structure of the blocks.

## Parameters

<i>argc</i>	The number of command-line arguments.
<i>argv</i>	The array of command-line arguments.

## Returns

An integer indicating the exit status of the program.

< Name of the CSV file to read.

< Name of the output file for blocks.

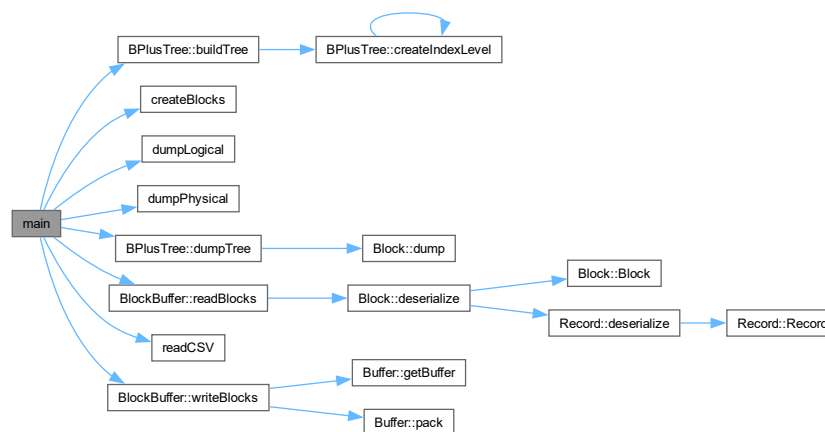
< Maximum records per block.

< Flag to indicate whether to dump physical structure.

< Flag to indicate whether to dump logical structure.

< [BlockBuffer](#) instance to handle file I/O operations.

< Vector to store the blocks. Here is the call graph for this function:



## 4.12.1.5 readCSV()

```
vector< string > readCSV (
    const string & filename)
```

Reads a CSV file (with a header) and returns a vector of CSV record strings.

Reads each line in the CSV file, skipping the header, and stores them in a vector.

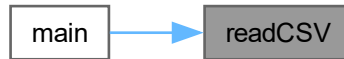
## Parameters

<i>filename</i>	The name of the CSV file to read.
-----------------	-----------------------------------

**Returns**

A vector of strings, each representing a record in the CSV file.

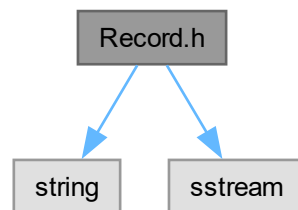
Here is the caller graph for this function:



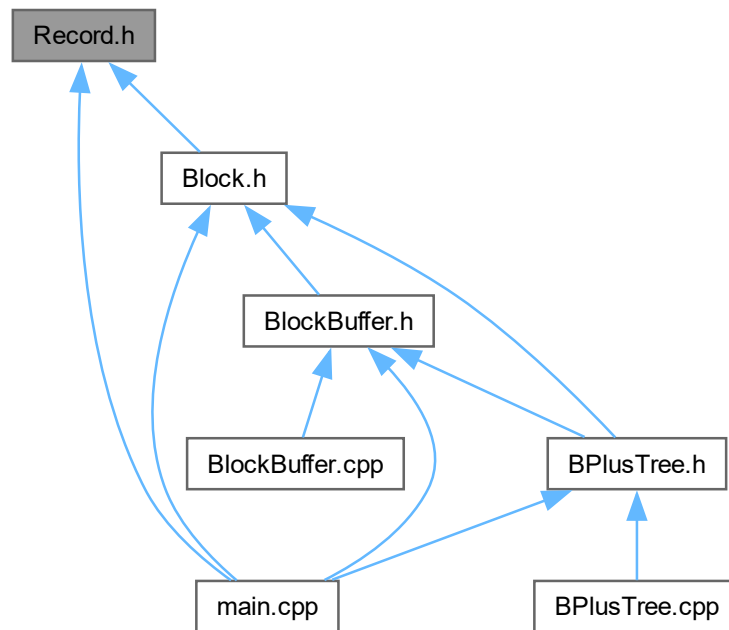
## 4.13 Record.h File Reference

```
#include <string>
#include <sstream>
```

Include dependency graph for Record.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [Record](#)

## 4.14 Record.h

[Go to the documentation of this file.](#)

```

00001 #ifndef RECORD_H
00002 #define RECORD_H
00003
00004 #include <string>
00005 #include <sstream>
00006
00007 class Record {
00008 public:
00009     int index;
00010     std::string field1;
00011     std::string field2;
00012     std::string field3;
00013
00014     Record() : index(0) {}
00015
00021     std::string serialize() const {
00022         std::stringstream ss;
00023         ss << index << "," << field1 << "," << field2 << "," << field3;
00024         return ss.str();
00025     }
00026
00033     static Record deserialize(const std::string &data) {
00034         Record r;
00035         std::stringstream ss(data);
00036         std::string token;
00037         getline(ss, token, ',');
  
```

```
00038         r.index = std::stoi(token);
00039         getline(ss, r.field1, ',');
00040         getline(ss, r.field2, ',');
00041         getline(ss, r.field3, ',');
00042         return r;
00043     }
00044 };
00045
00046 #endif
```

# Index

- Block, [5](#)
  - Block, [6](#)
  - blockNumber, [7](#)
  - blockType, [7](#)
  - deserialize, [6](#)
  - dump, [6](#)
  - nextBlock, [8](#)
  - records, [8](#)
  - serialize, [7](#)
- Block.h, [25](#)
  - BlockType, [26](#)
  - INDEX, [26](#)
  - LEAF, [26](#)
- BlockBuffer, [8](#)
  - BlockBuffer, [9](#)
  - filename, [12](#)
  - getFilename, [9](#)
  - readBlocks, [9](#)
  - setFilename, [10](#)
  - writeBlocks, [10](#)
- BlockBuffer.cpp, [28](#)
- BlockBuffer.h, [28](#)
- blockNumber
  - Block, [7](#)
- BlockType
  - Block.h, [26](#)
- blockType
  - Block, [7](#)
- BPlusTree, [12](#)
  - BPlusTree, [13](#)
  - buffer, [19](#)
  - buildTree, [14](#)
  - createIndexLevel, [14](#)
  - deleteRecord, [15](#)
  - dumpTree, [16](#)
  - filename, [19](#)
  - insert, [16](#)
  - mergeBlocks, [17](#)
  - recordsPerBlock, [19](#)
  - rootRBN, [19](#)
  - search, [17](#)
  - splitBlock, [18](#)
  - totalBlocks, [19](#)
- BPlusTree.cpp, [30](#)
- BPlusTree.h, [30](#)
- Buffer, [20](#)
  - buffer, [22](#)
  - getBuffer, [20](#)
  - pack, [20](#)
  - readHeader, [21](#)
  - unpack, [21](#)
  - writeHeader, [21](#)
- buffer
  - BPlusTree, [19](#)
  - Buffer, [22](#)
- Buffer.cpp, [32](#)
- Buffer.h, [33](#)
- buildTree
  - BPlusTree, [14](#)
- createBlocks
  - main.cpp, [35](#)
- createIndexLevel
  - BPlusTree, [14](#)
- deleteRecord
  - BPlusTree, [15](#)
- deserialize
  - Block, [6](#)
  - Record, [23](#)
- dump
  - Block, [6](#)
- dumpLogical
  - main.cpp, [35](#)
- dumpPhysical
  - main.cpp, [36](#)
- dumpTree
  - BPlusTree, [16](#)
- field1
  - Record, [23](#)
- field2
  - Record, [23](#)
- field3
  - Record, [24](#)
- filename
  - BlockBuffer, [12](#)
  - BPlusTree, [19](#)
- getBuffer
  - Buffer, [20](#)
- getFilename
  - BlockBuffer, [9](#)
- INDEX
  - Block.h, [26](#)
- index
  - Record, [24](#)
- insert
  - BPlusTree, [16](#)

- LEAF
  - Block.h, [26](#)
- main
  - main.cpp, [36](#)
- main.cpp, [34](#)
  - createBlocks, [35](#)
  - dumpLogical, [35](#)
  - dumpPhysical, [36](#)
  - main, [36](#)
  - readCSV, [37](#)
- mergeBlocks
  - BPlusTree, [17](#)
- nextBlock
  - Block, [8](#)
- pack
  - Buffer, [20](#)
- readBlocks
  - BlockBuffer, [9](#)
- readCSV
  - main.cpp, [37](#)
- readHeader
  - Buffer, [21](#)
- Record, [22](#)
  - deserialize, [23](#)
  - field1, [23](#)
  - field2, [23](#)
  - field3, [24](#)
  - index, [24](#)
  - Record, [22](#)
  - serialize, [23](#)
- Record.h, [38](#)
- records
  - Block, [8](#)
- recordsPerBlock
  - BPlusTree, [19](#)
- rootRBN
  - BPlusTree, [19](#)
- search
  - BPlusTree, [17](#)
- serialize
  - Block, [7](#)
  - Record, [23](#)
- setFilename
  - BlockBuffer, [10](#)
- splitBlock
  - BPlusTree, [18](#)
- totalBlocks
  - BPlusTree, [19](#)
- unpack
  - Buffer, [21](#)
- writeBlocks
  - BlockBuffer, [10](#)
- writeHeader
  - Buffer, [21](#)