BPlusTree

Generated by Doxygen 1.13.2

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

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

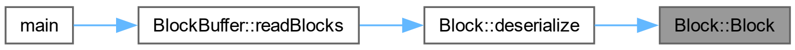
  List of records in this block.

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.

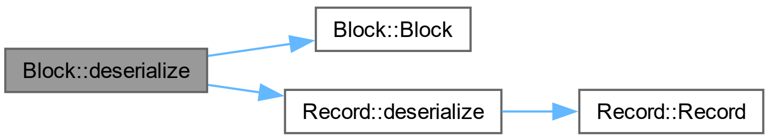
Parameters

data	The serialized block string.

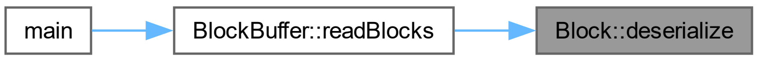
Returns

A Block object.

Here is the call graph for this function:



Here is the caller graph for this function:

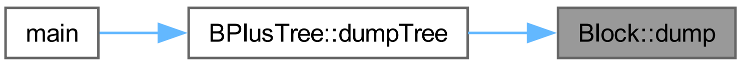


3.1.2.2 dump()

```
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]
```

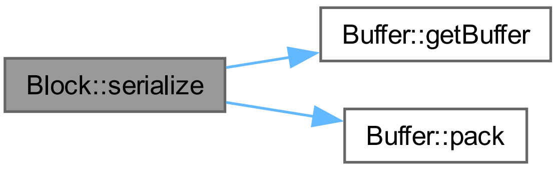
Serializes the block to a string.

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

file	The file name to initialize the BlockBuffer 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

filename	The input file name.
blocks	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.

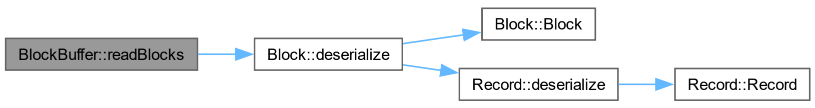
Parameters

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

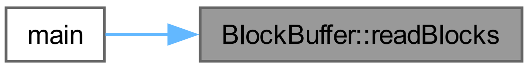
Returns

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

file	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

filename	The output file name.
blocks	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.

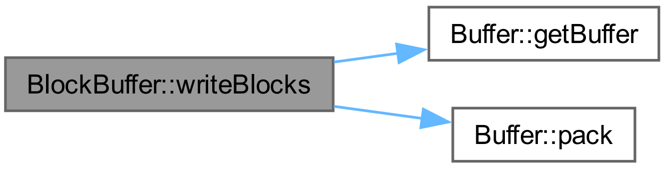
Parameters

filename	The name of the output file to write the blocks to.
blocks	A vector of Block objects to write to the file.

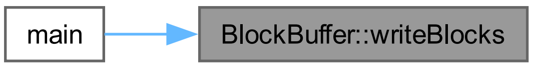
Returns

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:

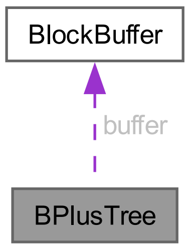
- BlockBuffer.h
- BlockBuffer.cpp

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

filename	The name of the file associated with the B+ Tree.
recordsPerBlock	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

filename	The name of the file associated with the B+ Tree.
recordsPerBlock	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

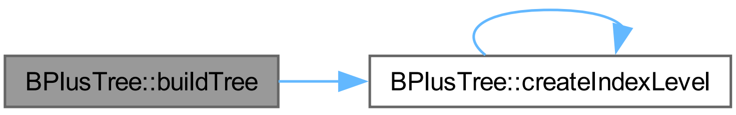
blocks	A vector of leaf blocks to form the base of the tree.

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.

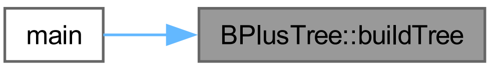
Parameters

blocks	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

keys	A vector of keys representing the largest keys from child blocks.
childRBNs	A vector of Relative Block 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.

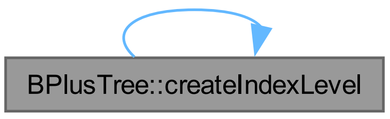
Parameters

keys	A vector of keys representing the largest keys from child blocks.
childRBNs	A vector of Relative Block Numbers (RBNs) for the child blocks.

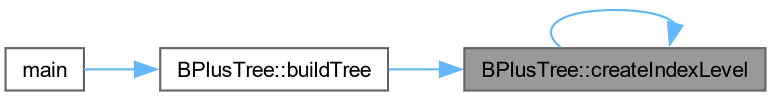
Returns

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

key	The key of the record to delete.

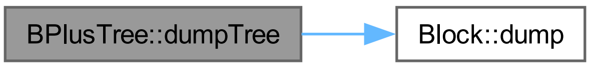
3.3.3.4 dumpTree()

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

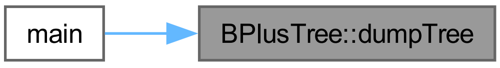
Dumps the structure of the B+ Tree for debugging purposes.

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

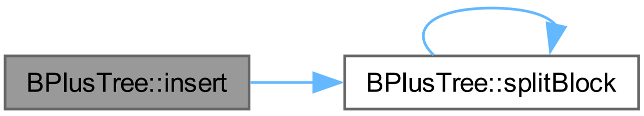
record	The record to be inserted into the tree.

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

record	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

leftBlock	The left sibling block.
rightBlock	The right sibling block.
parentRBN	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

key	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

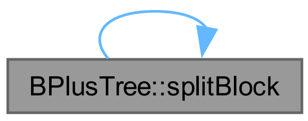
block	The overflowing block to split.
parentRBN	The RBN of the parent block, or -1 if the block has no parent.

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.

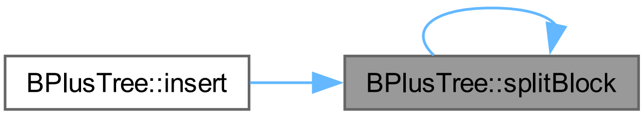
Parameters

block	The overflowing block that needs to be split.
parentRBN	The Relative Block 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()

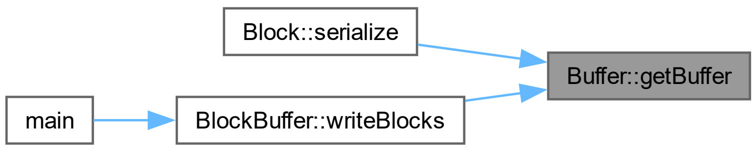
```
std::string Buffer::getBuffer () const  [inline]
```

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)
```

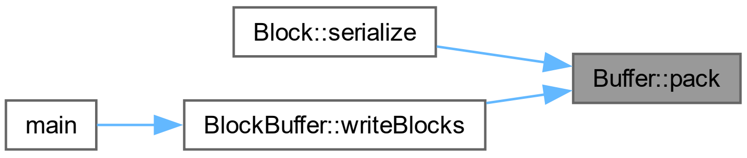
Packs a string into a length-indicated format.

Example: "Hello" becomes "5,Hello"

Parameters

data	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

file	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

file	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

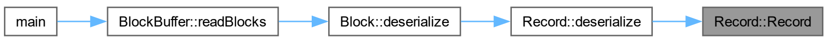
- int index
- std::string field1
- std::string field2
- std::string field3

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.

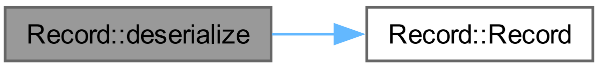
Parameters

data	The CSV string.

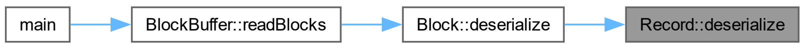
Returns

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

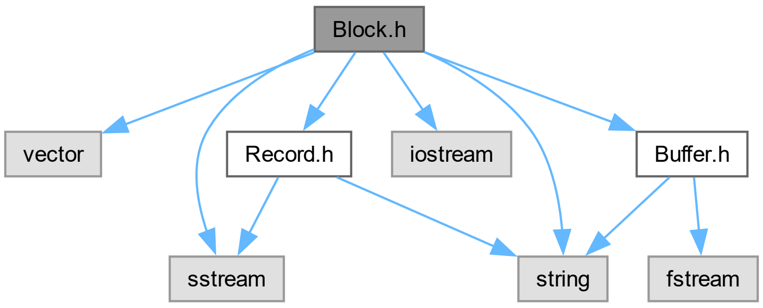
Chapter 4

File Documentation

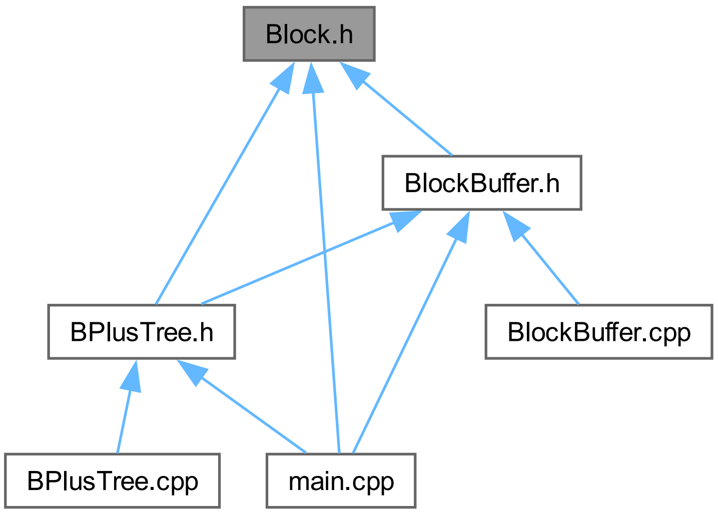
4.1 Block.h File Reference

```
#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

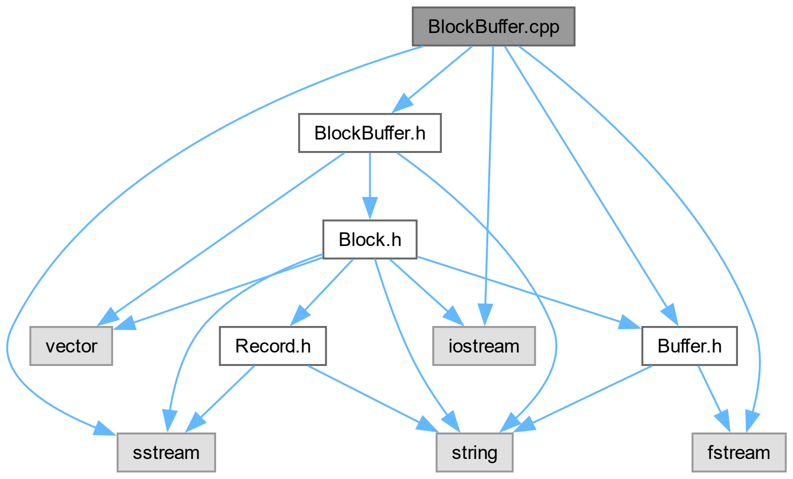
Go to the documentation of this file.

```
#ifndef BLOCK_H
#define BLOCK_H

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

// Enum for BlockType
enum class BlockType
{
    LEAF, // Leaf blocks contain records
    INDEX // Index blocks contain key-pointer pairs
};

class Block
{
public:
    int blockNumber;
    int nextBlock;
    BlockType blockType;
    std::vector<Record> records;

    // Default Constructor
    Block() : blockNumber(0), nextBlock(-1), blockType(BlockType::LEAF) {}

    std::string serialize() const
    {
        std::stringstream ss;
        // Write block header
        ss « blockNumber « "," « static_cast<int>(blockType) « "," « records.size() « "," « nextBlock « "\n";
        // Write each record (packed with Buffer)
        for (const auto &rec : records)
        {
            Buffer buf;
            std::string recStr = rec.serialize();
            buf.pack(recStr);
            ss « buf.getBuffer() « "\n";
        }
        return ss.str();
    }

    static Block deserialize(const std::string &data)
    {
        Block blk;
        std::stringstream ss(data);
        std::string line;
        // Get header line
        if (getline(ss, line))
        {
            std::stringstream headerStream(line);
            std::string token;
            getline(headerStream, token, ',');
            blk.blockNumber = std::stoi(token);
            getline(headerStream, token, ',');
            blk.blockType = static_cast<BlockType>(std::stoi(token)); // Convert int to BlockType
            getline(headerStream, token, ',');                        // Record count (not used here)
            getline(headerStream, token, ',');
            blk.nextBlock = std::stoi(token);
        }
        // Read each packed record
        while (getline(ss, line))
        {
            if (line.empty())
                continue;
            // Unpack the record manually
            size_t commaPos = line.find(',');
            if (commaPos == std::string::npos)
                continue;
            int len = std::stoi(line.substr(0, commaPos));
            std::string recData = line.substr(commaPos + 1, len);
            Record r = Record::deserialize(recData);
            blk.records.push_back(r);
        }
        return blk;
    }

    void dump() const
    {
        std::cout « "Block Number: " « blockNumber « ", Next Block: " « nextBlock
                  « ", Type: " « (blockType == BlockType::LEAF ? "LEAF" : "INDEX") « std::endl;
        std::cout « "Records:" « std::endl;
        for (const auto &r : records)
        {
            std::cout « r.index « " | " « r.field1 « " | " « r.field2 « " | " « r.field3 « std::endl;
        }
    }
};

#endif
```

4.3 BlockBuffer.cpp File Reference

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

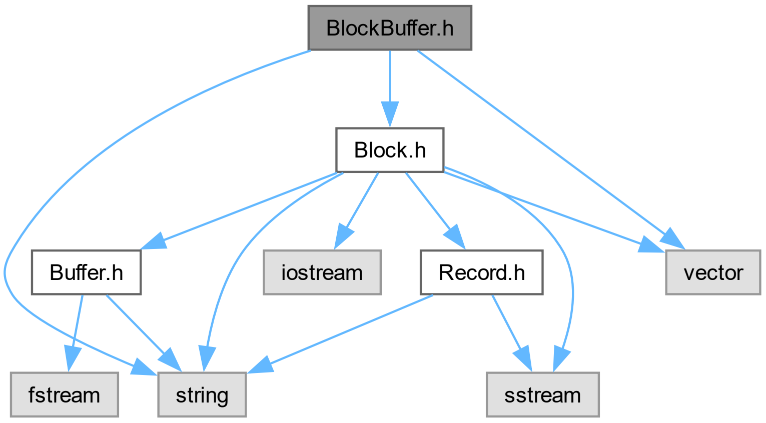
Include dependency graph for BlockBuffer.cpp:



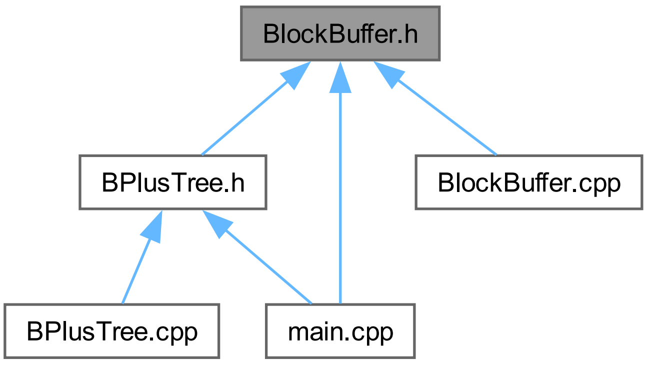
4.4 BlockBuffer.h File Reference

```
#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.

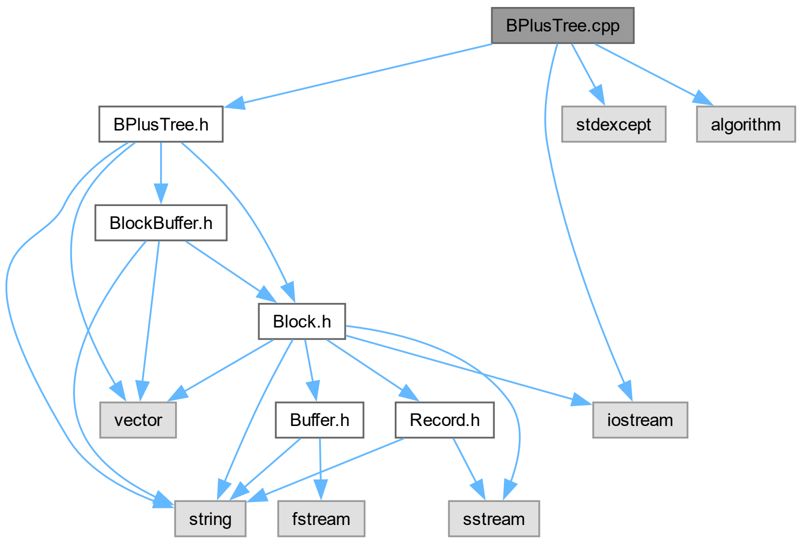
```
#ifndef BLOCKBUFFER_H
#define BLOCKBUFFER_H

```

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

class BlockBuffer
{
private:
    std::string filename; // Store the file name for file operations

public:
    BlockBuffer() = default;

    explicit BlockBuffer(const std::string &file) : filename(file) {}

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

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

    std::string getFilename() const { return filename; }

    void setFilename(const std::string &file) { filename = file; }
};

#endif
```

4.6 BPlusTree.cpp File Reference

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

Include dependency graph for BPlusTree.cpp:

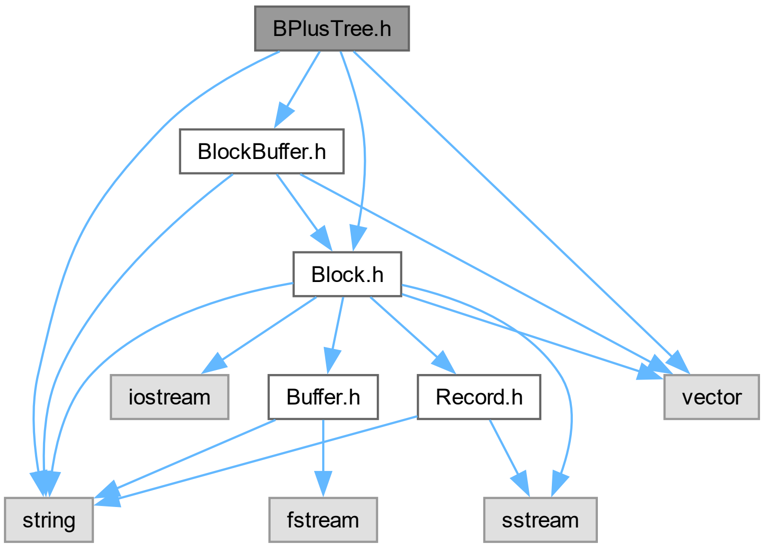


4.7 BPlusTree.h File Reference

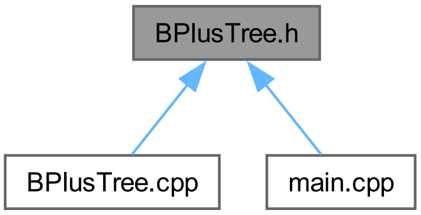
```
#include <string>
#include <vector>
#include "Block.h"
```

```
#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.

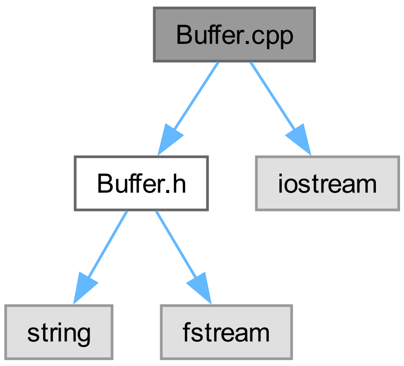
```
00001 #ifndef BPLUSTREE_H
```

```
#define BPLUSTREE_H

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

class BPlusTree
{
public:
    BPlusTree(const std::string &filename, int recordsPerBlock);

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

    void insert(const Record &record);

    void deleteRecord(int key);

    Record search(int key);

    void dumpTree();

private:
    // Metadata
    int rootRBN;
    int recordsPerBlock;
    int totalBlocks;
    std::string filename;

    // File handler
    BlockBuffer buffer;

    int createIndexLevel(const std::vector<int> &keys, const std::vector<int> &childRBNs);

    void splitBlock(Block &block, int parentRBN);

    void mergeBlocks(Block &leftBlock, Block &rightBlock, int parentRBN);
};

#endif
```

4.9 Buffer.cpp File Reference

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

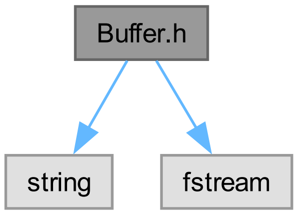
Include dependency graph for Buffer.cpp:



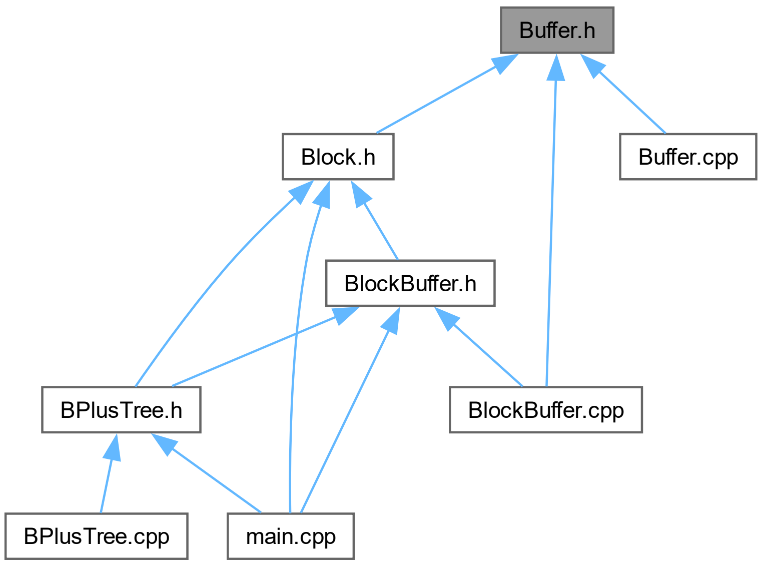
4.10 Buffer.h File Reference

```
#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

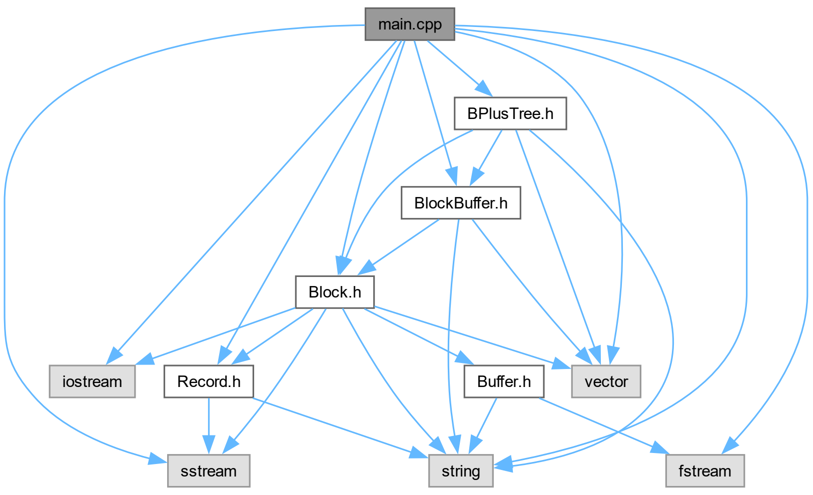
Go to the documentation of this file.

```
#ifndef BUFFER_H
#define BUFFER_H

#include <string>
#include <fstream>

class Buffer {
public:
    void pack(const std::string& data);

    std::string unpack();

    void readHeader(std::ifstream& file);

    void writeHeader(std::ofstream& file);

    std::string getBuffer() const { return buffer; }

private:
    std::string buffer;
};

#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.

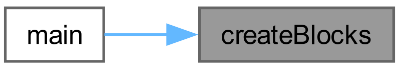
Parameters

records	A vector of CSV record strings to split into blocks.
recordsPerBlock	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)
```

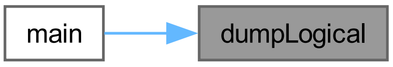
Dump blocks in logical order (following nextBlock pointer).

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

Parameters

blocks	A vector of Block objects to dump.

Here is the caller graph for this function:



4.12.1.3 dumpPhysical()

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

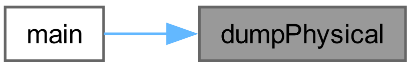
Dump blocks in physical order (as stored in file).

Iterates through the provided blocks and prints their contents sequentially.

Parameters

blocks	A vector of Block 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 the blocks.

Command-line flags:

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

Parameters

argc	The number of command-line arguments.
argv	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.

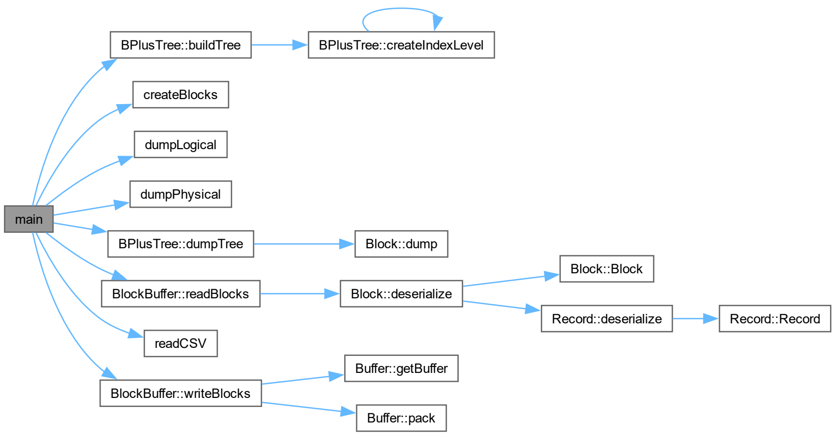
< Maximum records per block.

< 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.

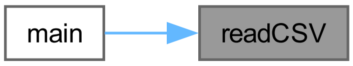
Parameters

filename	The name of the CSV file to read.

Returns

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

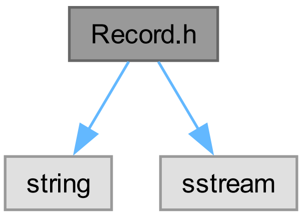
Here is the caller graph for this function:



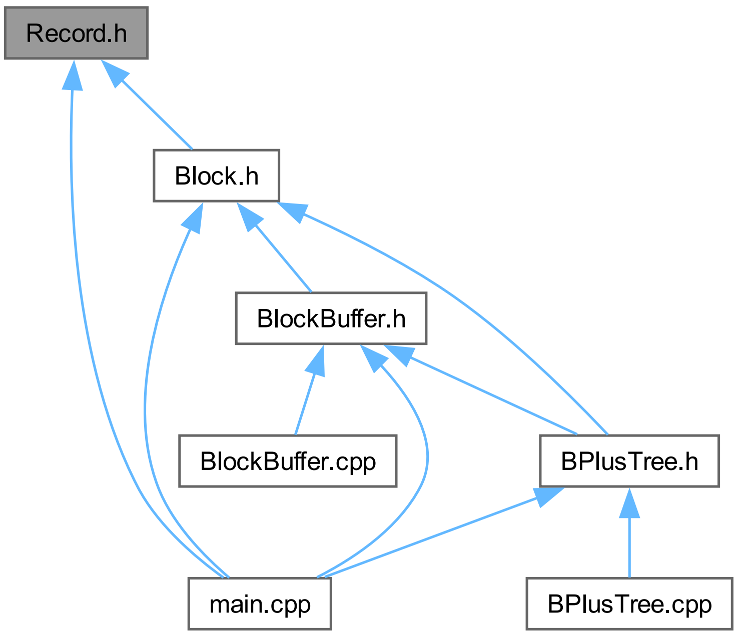
4.13 Record.h File Reference

```
#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.

```
#ifndef RECORD_H
#define RECORD_H

#include <string>
#include <sstream>

class Record {
public:
    int index;
    std::string field1;
    std::string field2;
    std::string field3;

    Record() : index(0) {}

    std::string serialize() const {
        std::stringstream ss;
        ss « index « "," « field1 « "," « field2 « "," « field3;
        return ss.str();
    }

    static Record deserialize(const std::string &data) {
        Record r;
        std::stringstream ss(data);
        std::string token;
        getline(ss, token, ',');
        r.index = std::stoi(token);
        getline(ss, r.field1, ',');
        getline(ss, r.field2, ',');
        getline(ss, r.field3, ',');
        return r;
    }
};

#endif
```

Index