Project

Generated by Doxygen 1.12.0

1 Class Index **1**

1.1 Class List 1

2 File Index **3**

2.1 File List 3

3 Class Documentation **5**

3.1 Buffer Class Reference 5

3.1.1 Detailed Description 6

3.1.2 Member Function Documentation 6

3.1.2.1 get_state_zip_codes() 6

3.1.2.2 parse_csv_line() 6

3.1.2.3 read_csv() 7

3.1.2.4 readLengthIndicatedRecord() 8

3.1.3 Member Data Documentation 8

3.1.3.1 records 8

3.2 CSVProcessing Class Reference 8

3.2.1 Detailed Description 9

3.2.2 Member Function Documentation 9

3.2.2.1 addHeader() 9

3.2.2.2 csvOutput() 9

3.2.2.3 sortBuffer() 10

3.3 FieldMetadata Struct Reference 11

3.3.1 Detailed Description 11

3.3.2 Member Data Documentation 11

3.3.2.1 name 11

3.3.2.2 typeSchema 11

3.4 HeaderBuffer Class Reference 12

3.4.1 Detailed Description 13

3.4.2 Constructor & Destructor Documentation 13

3.4.2.1 HeaderBuffer() 13

3.4.3 Member Function Documentation 14

3.4.3.1 addFieldMetadata() 14

3.4.3.2 getFieldCount() 14

3.4.3.3 getFields() 14

3.4.3.4 getFileStructureType() 15

3.4.3.5 getHeaderSize() 15

3.4.3.6 getIndexFileName() 16

3.4.3.7 getPrimaryKeyField() 16

3.4.3.8 getRecordCount() 16

3.4.3.9 getRecordSizeBytes() 17

3.4.3.10 getSizeFormat() 17

3.4.3.11 getVersion() 17

3.4.3.12 readHeader() 18
3.4.3.13 setFieldCount() 18
3.4.3.14 setFileStructureType() 19
3.4.3.15 setHeaderSize() 19
3.4.3.16 setIndexFileName() 19
3.4.3.17 setPrimaryKeyField() 20
3.4.3.18 setRecordCount() 20
3.4.3.19 setRecordSizeBytes() 21
3.4.3.20 setSizeFormat() 21
3.4.3.21 setVersion() 21
3.4.3.22 writeHeader() 22
3.4.4 Member Data Documentation 23
3.4.4.1 fieldCount 23
3.4.4.2 fields 23
3.4.4.3 fileStructureType 23
3.4.4.4 headerRecordSize 23
3.4.4.5 indexFileName 23
3.4.4.6 primaryKeyField 23
3.4.4.7 recordCount 23
3.4.4.8 recordSizeBytes 24
3.4.4.9 sizeFormatType 24
3.4.4.10 version 24
3.5 IndexFile Class Reference 24
3.5.1 Detailed Description 24
3.5.2 Member Function Documentation 25
3.5.2.1 createIndexFile() 25
3.6 ZipCodeRecord Struct Reference 25
3.6.1 Detailed Description 26
3.6.2 Member Data Documentation 26
3.6.2.1 city 26
3.6.2.2 latitude 26
3.6.2.3 longitude 26
3.6.2.4 state_id 26
3.6.2.5 zip_code 26

4 File Documentation **27**
4.1 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/buffer.cpp File Reference 27
4.1.1 Detailed Description 27
4.2 buffer.cpp 28
4.3 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/buffer.h File Reference 30
4.4 buffer.h 31
4.5 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/CSVLengthIndicated.cpp File Reference 31

4.5.1 Detailed Description 32
4.5.2 Function Documentation 33
4.5.2.1 convertCSVToLengthIndicated() 33
4.5.2.2 readLengthIndicatedRecord() 34
4.6 CSVLengthIndicated.cpp 35
4.7 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/CSVLengthIndicated.h File Reference 38
4.7.1 Detailed Description 39
4.7.2 Function Documentation 39
4.7.2.1 convertCSVToLengthIndicated() 39
4.7.2.2 readLengthIndicatedRecord() 41
4.8 CSVLengthIndicated.h 42
4.9 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/CSVProcessing.cpp File Reference 42
4.10 CSVProcessing.cpp 43
4.11 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/CSVProcessing.h File Reference 44
4.12 CSVProcessing.h 45
4.13 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/HeaderBuffer.cpp File Reference 46
4.14 HeaderBuffer.cpp 46
4.15 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/HeaderBuffer.h File Reference 48
4.16 HeaderBuffer.h 49
4.17 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/headerBufferTest.cpp File Reference 49
4.17.1 Function Documentation 50
4.17.1.1 main() 50
4.18 headerBufferTest.cpp 51
4.19 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/IndexFile.cpp File Reference . 52
4.20 IndexFile.cpp 53
4.21 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/IndexFile.h File Reference . . 54
4.22 IndexFile.h 54

Index **57**

Chapter 1

Class Index

1.1 Class List

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

Buffer 5
CSVProcessing 8
FieldMetadata
Structure to hold field metadata information 11
HeaderBuffer
Class for handling the data file header record 12
IndexFile 24
ZipCodeRecord 25

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/buffer.cpp
Implementation of the Buffer class and ZipCodeRecord struct 27
C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/buffer.h 30
C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/CSVLengthIndicated.cpp
Contains functions for converting a CSV file to a length-indicated format and reading length-indicated records 31
C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/CSVLengthIndicated.h
Header file for functions to handle length-indicated file conversion and reading 38
C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/CSVProcessing.cpp 42
C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/CSVProcessing.h 44
C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/HeaderBuffer.cpp 46
C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/HeaderBuffer.h 48
C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/headerBufferTest.cpp 49
C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/IndexFile.cpp 52
C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/IndexFile.h 54

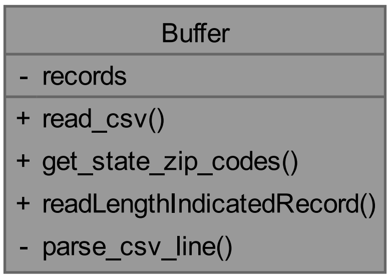
Chapter 3

Class Documentation

3.1 Buffer Class Reference

```
#include <buffer.h>
```

Collaboration diagram for Buffer:



Public Member Functions

- bool read_csv ()

  Reads the CSV file and stores the zip code records.
- std::map< std::string, std::vector< ZipCodeRecord > > get_state_zip_codes () const

  Groups the Zip Code records by state.
- bool readLengthIndicatedRecord (std::ifstream &fileStream, ZipCodeRecord &record)

  Reads a length-indicated record from a binary file.

Private Member Functions

- ZipCodeRecord parse_csv_line (const std::string &line) const

  Parses a line from the CSV into a ZipCodeRecord.

Private Attributes

- std::vector< ZipCodeRecord > records

3.1.1 Detailed Description

Definition at line 19 of file buffer.h.

3.1.2 Member Function Documentation

3.1.2.1 get_state_zip_codes()

```
std::map< std::string, std::vector< ZipCodeRecord > > Buffer::get_state_zip_codes () const
```

Groups the Zip Code records by state.

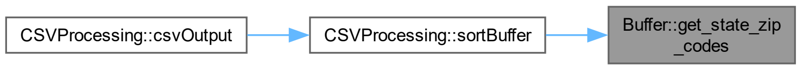
This function organizes the Zip Code records into a map where each state ID is a key, and the value is a vector of ZipCodeRecord structures associated with that state.

Returns

A map with state IDs as keys and vectors of ZipCodeRecord structures as values.

Definition at line 58 of file buffer.cpp.

Here is the caller graph for this function:



3.1.2.2 parse_csv_line()

```
ZipCodeRecord Buffer::parse_csv_line (
            const std::string & line) const  [private]
```

Parses a line from the CSV into a ZipCodeRecord.

This function takes a single line of CSV data and extracts the Zip Code, city, state ID, latitude, and longitude to populate a ZipCodeRecord structure.

Parameters

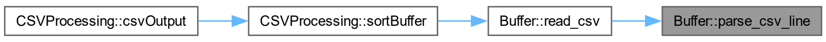
line	A string representing a single line from the CSV file.

Returns

A ZipCodeRecord structure containing the parsed data.

Definition at line 79 of file buffer.cpp.

Here is the caller graph for this function:



3.1.2.3 read_csv()

```
bool Buffer::read_csv ()
```

Reads the CSV file and stores the zip code records.

This function opens the CSV file, reads its contents, and parses each line into a ZipCodeRecord, which is stored in a vector.

Parameters

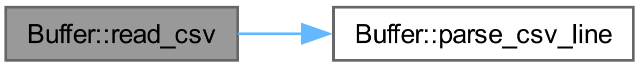
file_name	The path to the CSV file (us_postal_codes.csv).

Returns

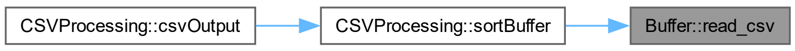
True if the file is read successfully, false otherwise.

Definition at line 28 of file buffer.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.4 readLengthIndicatedRecord()

```
bool Buffer::readLengthIndicatedRecord (
            std::ifstream & fileStream,
            ZipCodeRecord & record)
```

Reads a length-indicated record from a binary file.

This function reads a record from a length-indicated binary file, unpacks the fields, and stores them in a ZipCodeRecord.

Parameters

fileStream	The input binary file stream.
record	The ZipCodeRecord structure to populate.

Returns

True if a record is successfully read, false if end-of-file is reached or an error occurs.

Definition at line 130 of file buffer.cpp.

3.1.3 Member Data Documentation

3.1.3.1 records

```
std::vector<ZipCodeRecord> Buffer::records  [private]
```

Definition at line 32 of file buffer.h.

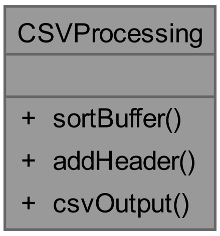
The documentation for this class was generated from the following files:

- C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/buffer.h
- C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/buffer.cpp

3.2 CSVProcessing Class Reference

```
#include <CSVProcessing.h>
```

Collaboration diagram for CSVProcessing:



Public Member Functions

- map< string, vector< ZipCodeRecord > > sortBuffer ()

  Sorts the CSV buffer and finds the zip codes (eastmost, westmost, northmost, southmost) for each state.
- void addHeader (string &file_name)

  Creates and adds a header to the CSV file.
- bool csvOutput (string &file_name)

  Outputs the processed zip code data to a CSV file.

3.2.1 Detailed Description

Definition at line 12 of file CSVProcessing.h.

3.2.2 Member Function Documentation

3.2.2.1 addHeader()

```
void CSVProcessing::addHeader (
             string & file_name)
```

Creates and adds a header to the CSV file.

This function adds a header row to the specified CSV file. The header includes columns for State, Easternmost, Westernmost, Northernmost, and Southernmost zip codes.

Parameters

file_name	The name of the CSV file to which the header will be added.

Definition at line 92 of file CSVProcessing.cpp.

3.2.2.2 csvOutput()

```
bool CSVProcessing::csvOutput (
             string & file_name)
```

Outputs the processed zip code data to a CSV file.

This function takes the sorted buffer of zip code records and writes them to a CSV file. Each row contains the state ID and the zip codes for the easternmost, westernmost, northernmost, and southernmost points in that state.

Parameters

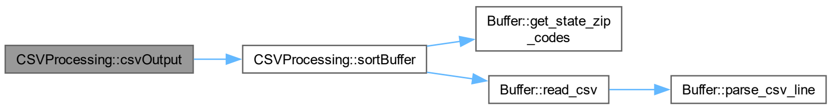
file_name	The name of the CSV file to which the data will be written.

Returns

true if the data was successfully written to the file, false otherwise.

Definition at line 112 of file CSVProcessing.cpp.

Here is the call graph for this function:



3.2.2.3 sortBuffer()

`std::map< string, std::vector< ZipCodeRecord > > CSVProcessing::sortBuffer ()`

Sorts the CSV buffer and finds the zip codes (eastmost, westmost, northmost, southmost) for each state.

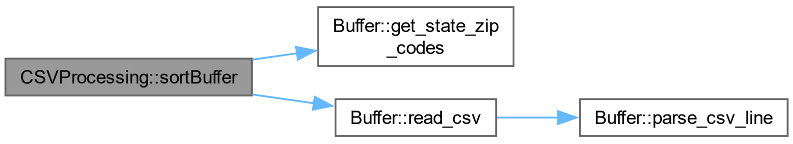
This method reads the CSV data, processes it to identify the easternmost, westernmost, northernmost, and southernmost zip codes for each state, and then stores these in a map (automatically sorts alphebetically).

Returns

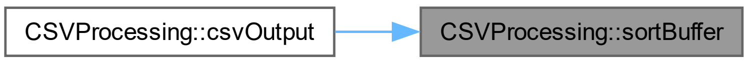
A map where the key is the state ID and the value is a vector containing the four ZipCodeRecord. The output looks as follows: [stateID] : { { east most zip, stateID, Cords }, { west most zip, stateID, Cords }, { northern most zip, stateID, Cords }, { southern most zip, stateID, Cords } }

Definition at line 32 of file CSVProcessing.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



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

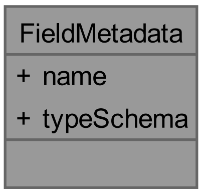
- C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/CSVProcessing.h
- C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/CSVProcessing.cpp

3.3 FieldMetadata Struct Reference

Structure to hold field metadata information.

```
#include <HeaderBuffer.h>
```

Collaboration diagram for FieldMetadata:



Public Attributes

- std::string name
- std::string typeSchema

3.3.1 Detailed Description

Structure to hold field metadata information.

Definition at line 12 of file HeaderBuffer.h.

3.3.2 Member Data Documentation

3.3.2.1 name

```
std::string FieldMetadata::name
```

Definition at line 13 of file HeaderBuffer.h.

3.3.2.2 typeSchema

```
std::string FieldMetadata::typeSchema
```

Definition at line 14 of file HeaderBuffer.h.

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

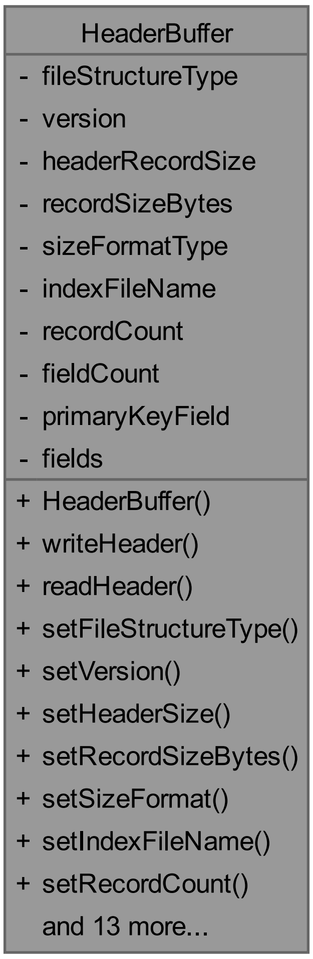
- C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/HeaderBuffer.h

3.4 HeaderBuffer Class Reference

Class for handling the data file header record.

`#include <HeaderBuffer.h>`

Collaboration diagram for HeaderBuffer:



Public Member Functions

- HeaderBuffer ()

  Default constructor for HeaderBuffer.
- bool writeHeader (const std::string &filename)

  Writes the header information to a file.
- bool readHeader (const std::string &filename)

  Reads and parses the header information from a file.
- void setFileStructureType (const std::string &type)

- void setVersion (const std::string &ver)
- void setHeaderSize (int size)
- void setRecordSizeBytes (int bytes)
- void setSizeFormat (const std::string &format)
- void setIndexFileName (const std::string &name)
- void setRecordCount (int count)
- void setFieldCount (int count)
- void setPrimaryKeyField (int field)
- void addFieldMetadata (const std::string &name, const std::string &schema)
- std::string getFileStructureType () const
- std::string getVersion () const
- int getHeaderSize () const
- int getRecordSizeBytes () const
- std::string getSizeFormat () const
- std::string getIndexFileName () const
- int getRecordCount () const
- int getFieldCount () const
- int getPrimaryKeyField () const
- const std::vector< FieldMetadata > & getFields () const

Private Attributes

- std::string fileStructureType
- std::string version
- int headerRecordSize
- int recordSizeBytes
- std::string sizeFormatType
- std::string indexFileName
- int recordCount
- int fieldCount
- int primaryKeyField
- std::vector< FieldMetadata > fields

3.4.1 Detailed Description

Class for handling the data file header record.

Definition at line 20 of file HeaderBuffer.h.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 HeaderBuffer()

```
HeaderBuffer::HeaderBuffer ()
```

Default constructor for HeaderBuffer.

Initializes all numeric members to zero

Definition at line 15 of file HeaderBuffer.cpp.

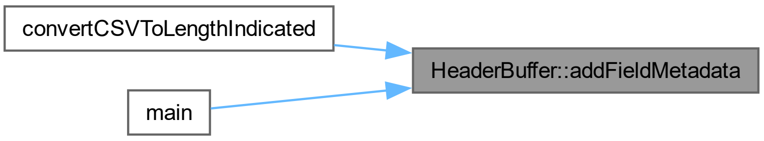
3.4.3 Member Function Documentation

3.4.3.1 addFieldMetadata()

```
void HeaderBuffer::addFieldMetadata (
            const std::string & name,
            const std::string & schema)  [inline]
```

Definition at line 51 of file HeaderBuffer.h.

Here is the caller graph for this function:

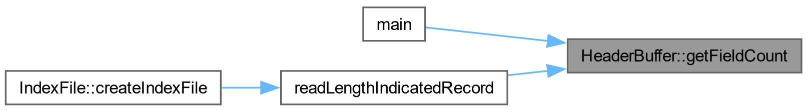


3.4.3.2 getFieldCount()

```
int HeaderBuffer::getFieldCount () const  [inline]
```

Definition at line 63 of file HeaderBuffer.h.

Here is the caller graph for this function:

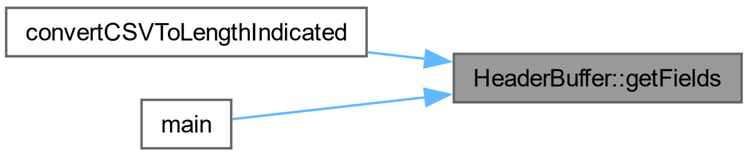

main
IndexFile::createIndexFile
readLengthIndicatedRecord
HeaderBuffer::getFieldCount

3.4.3.3 getFields()

```
const std::vector< FieldMetadata > & HeaderBuffer::getFields () const  [inline]
```

Definition at line 65 of file HeaderBuffer.h.

Here is the caller graph for this function:

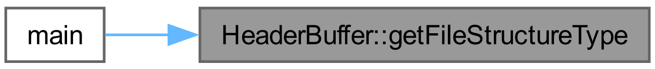


3.4.3.4 getFileStructureType()

```
std::string HeaderBuffer::getFileStructureType () const  [inline]
```

Definition at line 56 of file HeaderBuffer.h.

Here is the caller graph for this function:

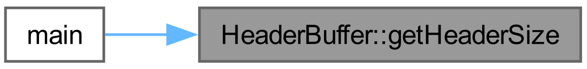


3.4.3.5 getHeaderSize()

```
int HeaderBuffer::getHeaderSize () const  [inline]
```

Definition at line 58 of file HeaderBuffer.h.

Here is the caller graph for this function:

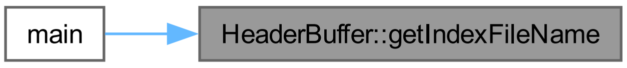


3.4.3.6 getIndexFileName()

```
std::string HeaderBuffer::getIndexFileName () const  [inline]
```

Definition at line 61 of file HeaderBuffer.h.

Here is the caller graph for this function:

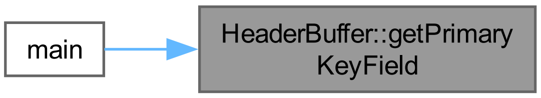


3.4.3.7 getPrimaryKeyField()

```
int HeaderBuffer::getPrimaryKeyField () const  [inline]
```

Definition at line 64 of file HeaderBuffer.h.

Here is the caller graph for this function:

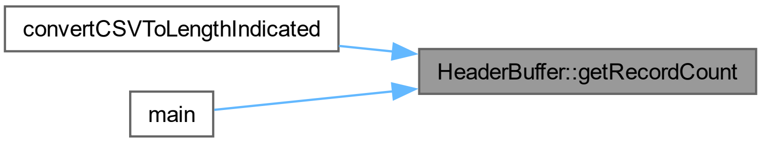


3.4.3.8 getRecordCount()

```
int HeaderBuffer::getRecordCount () const  [inline]
```

Definition at line 62 of file HeaderBuffer.h.

Here is the caller graph for this function:

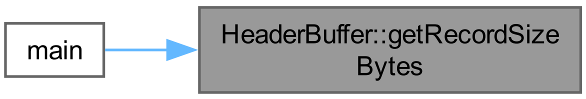


3.4.3.9 getRecordSizeBytes()

```
int HeaderBuffer::getRecordSizeBytes () const  [inline]
```

Definition at line 59 of file HeaderBuffer.h.

Here is the caller graph for this function:

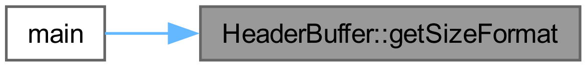


3.4.3.10 getSizeFormat()

```
std::string HeaderBuffer::getSizeFormat () const  [inline]
```

Definition at line 60 of file HeaderBuffer.h.

Here is the caller graph for this function:

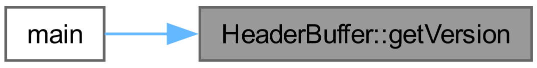


3.4.3.11 getVersion()

```
std::string HeaderBuffer::getVersion () const  [inline]
```

Definition at line 57 of file HeaderBuffer.h.

Here is the caller graph for this function:



3.4.3.12 readHeader()

```
bool HeaderBuffer::readHeader (
            const std::string & filename)
```

Reads and parses the header information from a file.

Parameters

filename	The name of the file to read from

Returns

true if the read operation was successful

false if there was an error opening or reading from the file

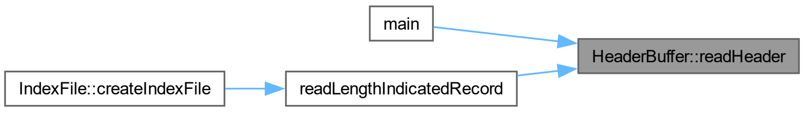
The method reads a header in length-indicated format where each field is preceded by a two-digit length indicator.

Exceptions

std::runtime_error	if the length indicators are invalid

Definition at line 88 of file HeaderBuffer.cpp.

Here is the caller graph for this function:

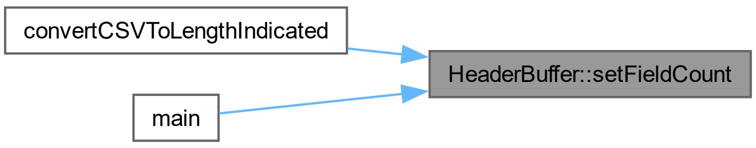


3.4.3.13 setFieldCount()

```
void HeaderBuffer::setFieldCount (
            int count) [inline]
```

Definition at line 49 of file HeaderBuffer.h.

Here is the caller graph for this function:

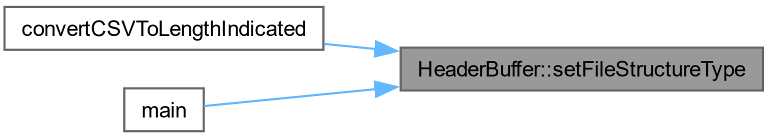


3.4.3.14 setFileStructureType()

```
void HeaderBuffer::setFileStructureType (
             const std::string & type)  [inline]
```

Definition at line 42 of file HeaderBuffer.h.

Here is the caller graph for this function:

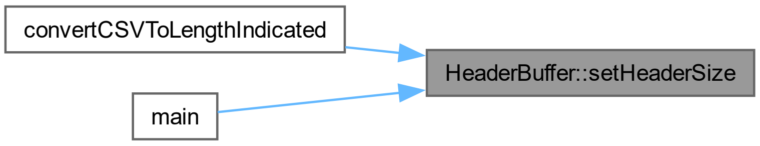

convertCSVToLengthIndicated
main
HeaderBuffer::setFileStructureType

3.4.3.15 setHeaderSize()

```
void HeaderBuffer::setHeaderSize (
             int size)  [inline]
```

Definition at line 44 of file HeaderBuffer.h.

Here is the caller graph for this function:

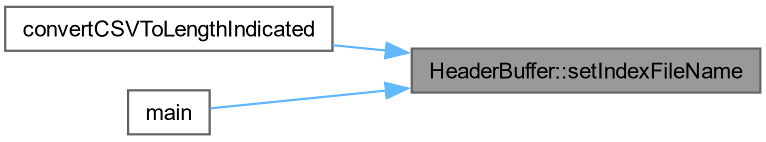


3.4.3.16 setIndexFileName()

```
void HeaderBuffer::setIndexFileName (
             const std::string & name)  [inline]
```

Definition at line 47 of file HeaderBuffer.h.

Here is the caller graph for this function:

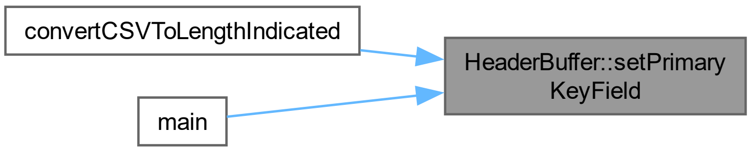


3.4.3.17 setPrimaryKeyField()

```
void HeaderBuffer::setPrimaryKeyField (
             int field)  [inline]
```

Definition at line 50 of file HeaderBuffer.h.

Here is the caller graph for this function:

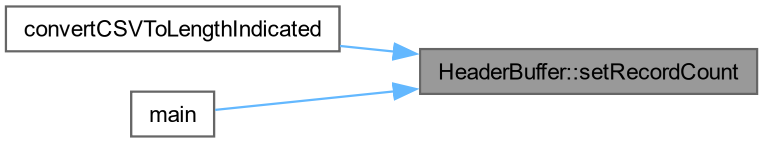

convertCSVToLengthIndicated
main
HeaderBuffer::setPrimary KeyField

3.4.3.18 setRecordCount()

```
void HeaderBuffer::setRecordCount (
             int count)  [inline]
```

Definition at line 48 of file HeaderBuffer.h.

Here is the caller graph for this function:

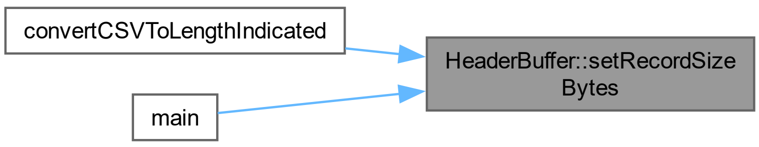


3.4.3.19 setRecordSizeBytes()

```
void HeaderBuffer::setRecordSizeBytes (
             int bytes)  [inline]
```

Definition at line 45 of file HeaderBuffer.h.

Here is the caller graph for this function:

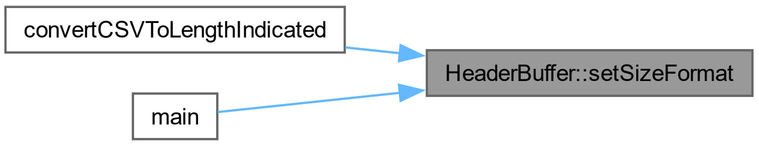

convertCSVToLengthIndicated
main
HeaderBuffer::setRecordSize Bytes

3.4.3.20 setSizeFormat()

```
void HeaderBuffer::setSizeFormat (
             const std::string & format)  [inline]
```

Definition at line 46 of file HeaderBuffer.h.

Here is the caller graph for this function:

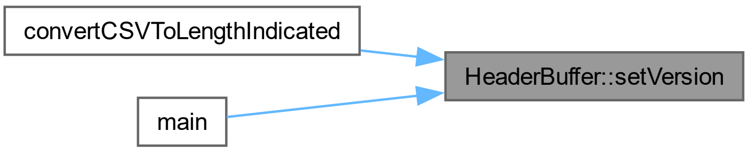


3.4.3.21 setVersion()

```
void HeaderBuffer::setVersion (
             const std::string & ver)  [inline]
```

Definition at line 43 of file HeaderBuffer.h.

Here is the caller graph for this function:



3.4.3.22 writeHeader()

```
bool HeaderBuffer::writeHeader (
             const std::string & filename)
```

Writes the header information to a file.

Parameters

filename	The name of the file to write to

Returns

true if the write operation was successful

false if there was an error opening or writing to the file

The header is written in a length-indicated format where each field is preceded by a two-digit length indicator. Fields are separated by commas. The header includes file structure information, version, sizes, and metadata about the fields in the file. Lambda function to write a length-indicated field

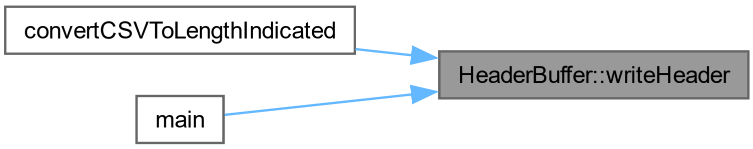
Parameters

value	The string value to write

Prepends a two-digit length indicator to the value

Definition at line 35 of file HeaderBuffer.cpp.

Here is the caller graph for this function:



3.4.4 Member Data Documentation

3.4.4.1 fieldCount

```
int HeaderBuffer::fieldCount  [private]
```

Definition at line 29 of file HeaderBuffer.h.

3.4.4.2 fields

```
std::vector<FieldMetadata> HeaderBuffer::fields  [private]
```

Definition at line 31 of file HeaderBuffer.h.

3.4.4.3 fileStructureType

```
std::string HeaderBuffer::fileStructureType  [private]
```

Definition at line 22 of file HeaderBuffer.h.

3.4.4.4 headerRecordSize

```
int HeaderBuffer::headerRecordSize  [private]
```

Definition at line 24 of file HeaderBuffer.h.

3.4.4.5 indexFileName

```
std::string HeaderBuffer::indexFileName  [private]
```

Definition at line 27 of file HeaderBuffer.h.

3.4.4.6 primaryKeyField

```
int HeaderBuffer::primaryKeyField  [private]
```

Definition at line 30 of file HeaderBuffer.h.

3.4.4.7 recordCount

```
int HeaderBuffer::recordCount  [private]
```

Definition at line 28 of file HeaderBuffer.h.

3.4.4.8 recordSizeBytes

```
int HeaderBuffer::recordSizeBytes  [private]
```

Definition at line 25 of file HeaderBuffer.h.

3.4.4.9 sizeFormatType

```
std::string HeaderBuffer::sizeFormatType  [private]
```

Definition at line 26 of file HeaderBuffer.h.

3.4.4.10 version

```
std::string HeaderBuffer::version  [private]
```

Definition at line 23 of file HeaderBuffer.h.

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

- C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/HeaderBuffer.h
- C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/HeaderBuffer.cpp

3.5 IndexFile Class Reference

```
#include <IndexFile.h>
```

Collaboration diagram for IndexFile:

Public Member Functions

- bool createIndexFile (const std::string &csvFile, const std::string &outputFile)

3.5.1 Detailed Description

Definition at line 8 of file IndexFile.h.

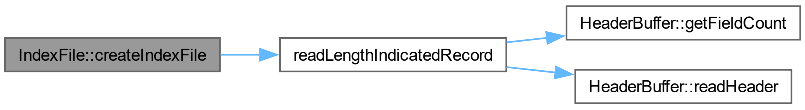
3.5.2 Member Function Documentation

3.5.2.1 createIndexFile()

```
bool IndexFile::createIndexFile (
            const std::string & csvFile,
            const std::string & outputFile)
```

Definition at line 10 of file IndexFile.cpp.

Here is the call graph for this function:


IndexFile::createIndexFile
readLengthIndicatedRecord
HeaderBuffer::getFieldCount
HeaderBuffer::readHeader

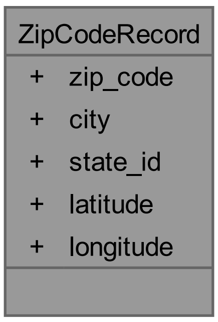
The documentation for this class was generated from the following files:

- C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/IndexFile.h
- C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/IndexFile.cpp

3.6 ZipCodeRecord Struct Reference

```
#include <buffer.h>
```

Collaboration diagram for ZipCodeRecord:



Public Attributes

- std::string zip_code
- std::string city
- std::string state_id
- double latitude
- double longitude

3.6.1 Detailed Description

Definition at line 10 of file buffer.h.

3.6.2 Member Data Documentation

3.6.2.1 city

```
std::string ZipCodeRecord::city
```

Definition at line 12 of file buffer.h.

3.6.2.2 latitude

```
double ZipCodeRecord::latitude
```

Definition at line 14 of file buffer.h.

3.6.2.3 longitude

```
double ZipCodeRecord::longitude
```

Definition at line 15 of file buffer.h.

3.6.2.4 state_id

```
std::string ZipCodeRecord::state_id
```

Definition at line 13 of file buffer.h.

3.6.2.5 zip_code

```
std::string ZipCodeRecord::zip_code
```

Definition at line 11 of file buffer.h.

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

- C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/buffer.h

Chapter 4

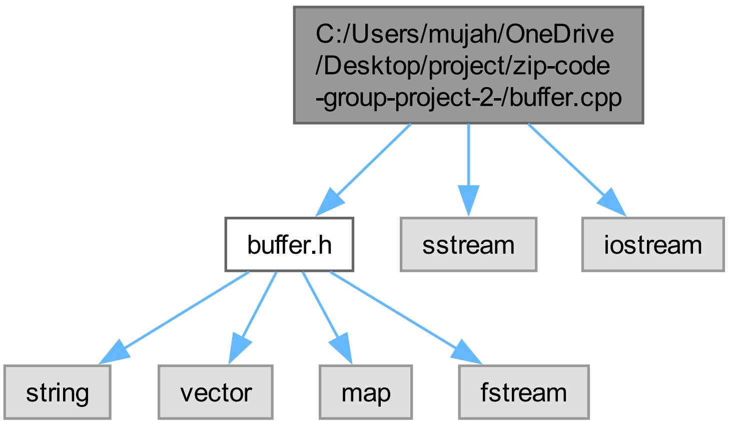
File Documentation

4.1 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/buffer.cpp File Reference

Implementation of the Buffer class and ZipCodeRecord struct.

```
#include "buffer.h"
#include <sstream>
#include <iostream>
```

Include dependency graph for buffer.cpp:


C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/buffer.cpp
buffer.h
sstream
iostream
string
vector
map
fstream

4.1.1 Detailed Description

Implementation of the Buffer class and ZipCodeRecord struct.

Implementation of the Buffer class for handling Zip Code data read from the CSV file us_postal_codes.csv.

Author

Daniel Eze

Date

9/29/2024

Definition in file buffer.cpp.

4.2 buffer.cpp

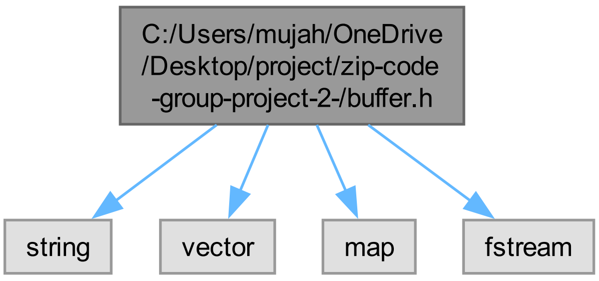
Go to the documentation of this file.

```
// Buffer.cpp
#include "buffer.h"
#include <sstream>
#include <iostream>

bool Buffer::read_csv() {
    std::ifstream file("us_postal_codes.csv"); // Open the file
    if (!file.is_open()) {
        std::cerr « "Error opening file: us_postal_codes.csv" « std::endl;
        return false;
    }

    std::string line;
    std::getline(file, line); // Skip the header line

    // Read each line of the file
    while (std::getline(file, line)) {
        records.push_back(parse_csv_line(line)); // Parse and store the line
    }

    file.close(); // Close the file
    std::cout « "CSV is now in the buffer" « std::endl;
    return true; // Return true if reading was successful
}

std::map<std::string, std::vector<ZipCodeRecord» Buffer::get_state_zip_codes() const {
    std::map<std::string, std::vector<ZipCodeRecord» state_zip_map; // Create a map to hold state records

    // Loop through all records
    for (const auto& record : records) {
        state_zip_map[record.state_id].push_back(record); // Add record to the correct state
    }

    return state_zip_map; // Return the grouped records
}

ZipCodeRecord Buffer::parse_csv_line(const std::string& line) const {
    std::stringstream ss(line); // Use stringstream to parse the line
    ZipCodeRecord record; // Create a ZipCodeRecord to hold the data
    std::string skip;

    // Extract and store each field
    std::getline(ss, record.zip_code, ','); // Get Zip Code
    std::getline(ss, record.city, ',');     // Get City
    std::getline(ss, record.state_id, ','); // Get State ID
    std::getline(ss, skip, ',');            // Skip a field
    std::string latitude_str, longitude_str;
    std::getline(ss, latitude_str, ',');    // Get Latitude as string
    std::getline(ss, longitude_str, ',');   // Get Longitude as string

    try {
        if (!latitude_str.empty()) {
            record.latitude = std::stod(latitude_str); // Convert to double
        } else {
            std::cerr « "Invalid latitude value for Zip Code: " « record.zip_code « std::endl;
            record.latitude = 0.0; // Default value or handle appropriately
        }

        if (!longitude_str.empty()) {
            record.longitude = std::stod(longitude_str); // Convert to double
        } else {
            std::cerr « "Invalid longitude value for Zip Code: " « record.zip_code « std::endl;
            record.longitude = 0.0; // Default value or handle appropriately
        }
    } catch (const std::invalid_argument& e) {
        std::cerr « "Error: Invalid numeric value in CSV for Zip Code: " « record.zip_code « " " «
record.state_id « std::endl;
        record.latitude = 0.0;  // Default value or handle appropriately
        record.longitude = 0.0; // Default value or handle appropriately
    } catch (const std::out_of_range& e) {
        std::cerr « "Error: Out of range numeric value in CSV for Zip Code: " « record.zip_code «
std::endl;
        record.latitude = 0.0;  // Default value or handle appropriately
        record.longitude = 0.0; // Default value or handle appropriately
    }

    return record; // Return the populated record
}

bool Buffer::readLengthIndicatedRecord( std::ifstream& fileStream, ZipCodeRecord& record ) {
    if ( !fileStream.is_open() || fileStream.eof() ) {
        return false;
    }

    std::string line;
    if ( !std::getline( fileStream, line ) ) {
        return false;  // EOF reached
    }

    std::stringstream ss( line );

    auto parseField = [ ]( std::stringstream& ss ) -> std::string {
        // Skip any leading whitespace
        while ( std::isspace( ss.peek() ) ) {
            ss.get();
        }

        // Read the two-digit length
        char lengthChars[ 3 ] = { 0 };  // Two digits + null terminator
        if ( !ss.read( lengthChars, 2 ) ) {
            throw std::runtime_error( "Failed to read field length." );
        }

        std::string lengthStr( lengthChars, 2 );  // Ensure we have exactly 2 characters
        int fieldLength = 0;

        try {
            fieldLength = std::stoi( lengthStr );
        }
        catch ( const std::exception& e ) {
            std::cerr « "Invalid field length: '" « lengthStr « "'" « std::endl;
            throw;
        }

        // Read the field data
        std::string fieldData( fieldLength, '\0' );
        if ( !ss.read( &fieldData[ 0 ], fieldLength ) ) {
            throw std::runtime_error( "Failed to read field data." );
        }

        // Check if we've read the expected number of characters
        if ( fieldData.length() != static_cast< size_t >( fieldLength ) ) {
            throw std::runtime_error( "Field data length mismatch." );
        }

        // Consume the comma delimiter if not at the end
        if ( ss.peek() == ',' ) {
            ss.get();
        }

        return fieldData;
        };

    try {
        // Parse all fields, including 'County'
        record.zip_code = parseField( ss );        // Field 1: Zip Code
        record.city = parseField( ss );            // Field 2: City
        record.state_id = parseField( ss );        // Field 3: State ID
        std::string county = parseField( ss );     // Field 4: County
        std::string latitude_str = parseField( ss );   // Field 5: Latitude
        std::string longitude_str = parseField( ss );  // Field 6: Longitude

        // Convert latitude and longitude from string to double
        record.latitude = std::stod( latitude_str );
        record.longitude = std::stod( longitude_str );
    }
    catch ( const std::exception& e ) {
        std::cerr « "Error parsing record: " « e.what() « std::endl;
        return false;
    }

    return true;
}
```

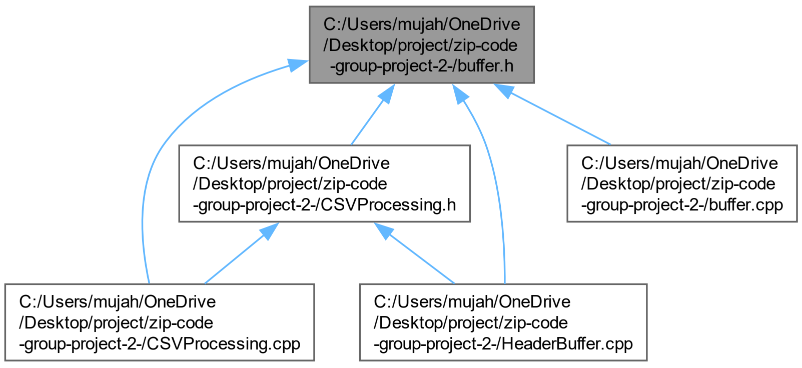
4.3 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/buffer.h File Reference

```
#include <string>
#include <vector>
#include <map>
#include <fstream>
```

Include dependency graph for buffer.h:



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


C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/buffer.h
C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/CSVProcessing.h
C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/buffer.cpp
C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/CSVProcessing.cpp
C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/HeaderBuffer.cpp

Classes

- struct ZipCodeRecord
- class Buffer

4.4 buffer.h

Go to the documentation of this file.

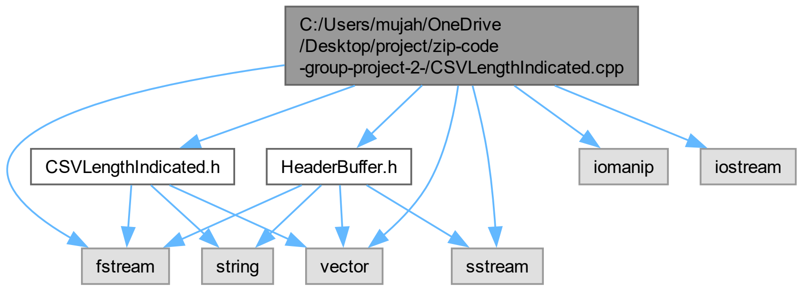
```
#ifndef BUFFER_H
#define BUFFER_H

#include <string>
#include <vector>
#include <map>
#include <fstream>

// Define ZipCodeRecord structure
struct ZipCodeRecord {
    std::string zip_code;
    std::string city;
    std::string state_id;
    double latitude;
    double longitude;
};

// Define Buffer class
class Buffer {
public:
    // Method to read a CSV file and store records
    bool read_csv();

    // Method to get records grouped by state
    std::map<std::string, std::vector<ZipCodeRecord» get_state_zip_codes() const;

    // Method to read and unpack a length-indicated Zip Code record
    bool readLengthIndicatedRecord(std::ifstream &fileStream, ZipCodeRecord &record);

private:
    // Vector to store ZipCodeRecord entries
    std::vector<ZipCodeRecord> records;

    // Method to parse a line from CSV into ZipCodeRecord
    ZipCodeRecord parse_csv_line(const std::string& line) const;
};

#endif
```

4.5 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/CSVLengthIndicated.cpp File Reference

Contains functions for converting a CSV file to a length-indicated format and reading length-indicated records.

```
#include "CSVLengthIndicated.h"
#include "HeaderBuffer.h"
#include <fstream>
#include <sstream>
#include <iomanip>
#include <iostream>
#include <vector>
```

Include dependency graph for CSVLengthIndicated.cpp:



Functions

- void convertCSVToLengthIndicated (const std::string &csvFileName, const std::string &outputFileName)

    Converts a CSV file to a length-indicated format.
- std::vector< std::vector< std::string > > readLengthIndicatedRecord (const std::string &filename)

    Reads data from a CSV file, skipping the header row.

4.5.1 Detailed Description

Contains functions for converting a CSV file to a length-indicated format and reading length-indicated records.

This file provides the implementation of functions that convert a CSV file to a length-indicated format and read records from a CSV file. The length-indicated format is a custom representation where each field is prefixed by its length, allowing for variable-length records.

The provided functions include:

- `convertCSVToLengthIndicated()`: Converts the data in a CSV file to a length-indicated format.
- `readCSV()`: Reads a CSV file, ignoring the header row, and returns data as a vector of rows.

Note

Length-indicated records are written as plain text, with each field prefixed by its length as a two-digit integer.

Author

Thomas Hoerger

Date

October 18 2024

Definition in file CSVLengthIndicated.cpp.

4.5.2 Function Documentation

4.5.2.1 convertCSVToLengthIndicated()

```
void convertCSVToLengthIndicated (
             const std::string & csvFileName,
             const std::string & outputFileName)
```

Converts a CSV file to a length-indicated format.

Reads each record from a CSV file and writes it to an output file, where each field in the record is prefixed by its length as a two-digit integer. The header row is written without length indicators.

Parameters

csvFileName	The name of the CSV file to be converted.
outputFileName	The name of the output file where length-indicated records will be stored.

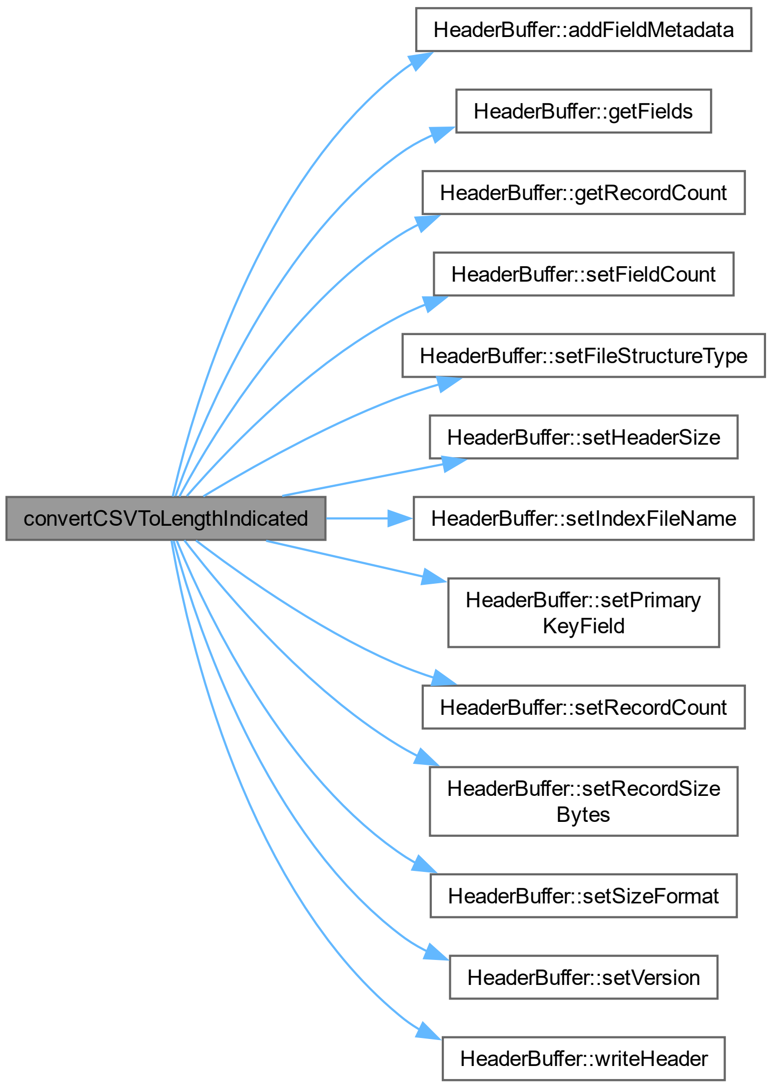
Note

Each field's length is formatted as a two-digit number, padded with zeroes if necessary.

If a field exceeds 99 characters, it will be truncated to fit within the two-digit length limit.

Definition at line 44 of file CSVLengthIndicated.cpp.

Here is the call graph for this function:


convertCSVToLengthIndicated
HeaderBuffer::addFieldMetadata
HeaderBuffer::getFields
HeaderBuffer::getRecordCount
HeaderBuffer::setFieldCount
HeaderBuffer::setFileStructureType
HeaderBuffer::setHeaderSize
HeaderBuffer::setIndexFileName
HeaderBuffer::setPrimary KeyField
HeaderBuffer::setRecordCount
HeaderBuffer::setRecordSize Bytes
HeaderBuffer::setSizeFormat
HeaderBuffer::setVersion
HeaderBuffer::writeHeader

4.5.2.2 readLengthIndicatedRecord()

```
std::vector< std::vector< std::string > > readLengthIndicatedRecord (
            const std::string & filename)
```

Reads data from a CSV file, skipping the header row.

Reads a length-indicated record from a file stream.

Reads each row of a CSV file into a vector of strings, where each inner vector represents a row in the CSV. The header row is ignored, and only data rows are returned.

Parameters

filename	The name of the CSV file to be read.

Returns

A vector of vectors, where each inner vector represents a row of fields.

Note

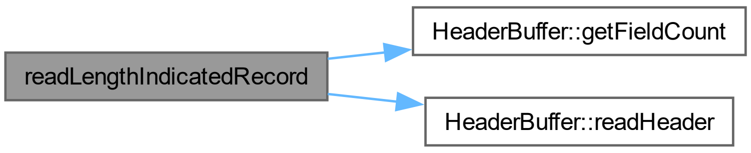
This function assumes a simple CSV structure with fields separated by commas.

Warning

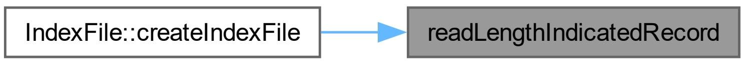
An error message is displayed if the file cannot be opened.

Definition at line 182 of file CSVLengthIndicated.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6 CSVLengthIndicated.cpp

Go to the documentation of this file.

```
#include "CSVLengthIndicated.h"
#include "HeaderBuffer.h"
#include <fstream>
#include <sstream>
#include <iomanip>
```

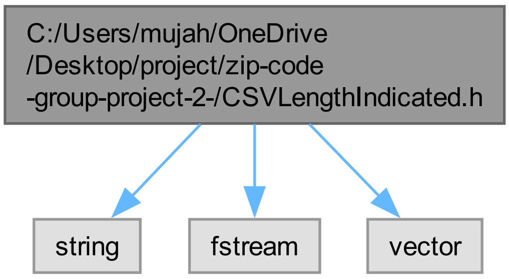
```
#include <iostream>  // Added this for std::cerr
#include <vector>
#include "HeaderBuffer.h"


void convertCSVToLengthIndicated(const std::string& csvFileName, const std::string& outputFileName) {
    std::ifstream inputFile(csvFileName);  // Open the CSV file for reading
    std::ofstream outputFile(outputFileName);  // Open the output file for writing
    HeaderBuffer header;


    HeaderBuffer header;
    header.setFileStructureType("CSV");
    header.setVersion("1.0");
    header.setHeaderSize(256);
    header.setRecordSizeBytes(4);
    header.setSizeFormat("ASCII");
    header.setIndexFileName("headerBufferTest.csv");
    header.setRecordCount(1000);
    header.setFieldCount(6);
    header.setPrimaryKeyField(0);
    header.writeHeader( csvFileName );
    // Check if either file failed to open
    if (!inputFile.is_open() || !outputFile.is_open()) {
        std::cerr « "Failed to open file(s)." « std::endl;
        return;
    }

    // Initialize header properties
    header.setFileStructureType("CSV");
    header.setVersion("1.0");
    header.setSizeFormat("2D"); // Two-digit length indicators
    header.setIndexFileName("index.txt"); // No index file for now

    std::string headerLine;
    std::getline(inputFile, headerLine);

    // Parse header to count fields and determine field metadata
    std::istringstream headerStream(headerLine);
    std::string fieldName;
    while (std::getline(headerStream, fieldName, ',')) {
        // Remove any quotation marks from field names
        if (!fieldName.empty() && fieldName.front() == '"' && fieldName.back() == '"') {
            fieldName = fieldName.substr(1, fieldName.length() - 2);
        }

        FieldMetadata field;
        field.name = fieldName;
        field.typeSchema = "STRING"; // Default type
        header.addFieldMetadata(fieldName, "string");
    }

    header.setFieldCount(header.getFields().size());
    header.setPrimaryKeyField(0); // Assume first field is primary key

    std::string line;
    size_t maxRecordSize = 0;
    header.setRecordCount(0);

    while (std::getline(inputFile, line)) {
        header.setRecordCount(header.getRecordCount() + 1);
        maxRecordSize = std::max(maxRecordSize, line.length());
    }

    header.setRecordSizeBytes(maxRecordSize);
    header.setHeaderSize(headerLine.length());

    // Write header
    header.writeHeader(outputFileName);

    // Reset file position to start of data
    inputFile.clear();
    inputFile.seekg(0);
    std::getline(inputFile, line); // Skip header line again

    // Process each line in the CSV file
    while (std::getline(inputFile, line)) {
        // Skip the header row and write it as-is without length indicators
        // if (isFirstRow) {
        //     isFirstRow = false;  // Set the flag to false after processing the header
        //     continue;
        // }

        std::istringstream ss(line);  // String stream to parse each field in the line
        std::string token;
        std::string lengthIndicatedLine;
        bool isFirstToken = true;  // Flag for adding commas between fields

        // Process each comma-separated field in the line
        while (std::getline(ss, token, ',')) {
            // Add a comma before each field except the first one
            if (!isFirstToken) {
                lengthIndicatedLine += ",";
            }

            // Remove enclosing quotation marks if they exist
            if (!token.empty() && token.front() == '"' && token.back() == '"') {
                token = token.substr(1, token.length() - 2);
            }

            // Limit the field length to 99 characters, and log a warning if truncated
            if (token.length() > 99) {
                std::cerr « "Field length exceeds two-digit limit: " « token « std::endl;
                token = token.substr(0, 99);
            }

            // If the field contains a decimal, format it as a fixed-precision floating-point number
            if (token.find('.') != std::string::npos && (isdigit(token[0]) || token[0] == '-')) {
                double num = std::stod(token);  // Convert the string to a double
                std::ostringstream oss;
                oss « std::fixed « std::setprecision(6) « num;  // Format with fixed precision
                token = oss.str();
            }

            int fieldLength = token.length();  // Calculate the field length

            // Create a formatted string with the field length followed by the field value
            std::stringstream lengthToken;
            lengthToken « std::setw(2) « std::setfill('0') « fieldLength « token;
            lengthIndicatedLine += lengthToken.str();  // Append the formatted field to the output
line

            isFirstToken = false;  // Set flag to false after the first token
        }

        outputFile « lengthIndicatedLine « std::endl;  // Write the formatted line to the output file
    }

    inputFile.close();  // Close the input file
    outputFile.close();  // Close the output file
}

std::vector<std::vector<std::string» readLengthIndicatedRecord( const std::string& filename ) {
    std::vector<std::vector<std::string» data;  // Outer vector to store all rows
    HeaderBuffer header;

    // Read the header first
    if (!header.readHeader(filename)) {
        std::cerr « "Error: Could not read header from " « filename « std::endl;
        return data;
    }

    std::ifstream file(filename);  // Open the CSV file for reading
    if (!file.is_open()) {  // Check if the file failed to open
        std::cerr « "Error: Could not open file " « filename « std::endl;
        return data;
    }

    std::string line;
    size_t recordsRead = 0;

    // Skip past the header section
    while (std::getline(file, line) && recordsRead < header.getFieldCount() + 1) {
        recordsRead++;
    }
    //bool isHeader = true;  // Flag to skip the header row

    // Read each line from the file
    while (std::getline(file, line)) {
        // if (isHeader) {  // Skip the header row
        //     isHeader = false;
        //     continue;
        // }

        std::istringstream ss(line);  // String stream for parsing the line
        std::vector<std::string> row;  // Inner vector to store fields in each row
        std::string field;

        // Parse each field separated by commas
        while (std::getline(ss, field, ',')) {
            row.push_back(field);  // Add the field to the row vector
        }

        data.push_back(row);  // Add the row vector to the data vector
    }

    file.close();  // Close the file
    return data;   // Return the populated data
}
```

4.7 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/CSVLengthIndicated.h File Reference

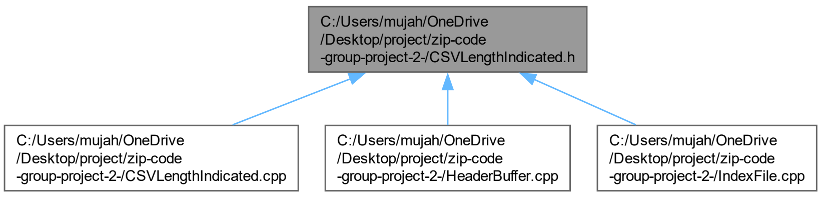
Header file for functions to handle length-indicated file conversion and reading.

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

Include dependency graph for CSVLengthIndicated.h:


C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/CSVLengthIndicated.h
string
fstream
vector

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


C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/CSVLengthIndicated.h
C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/CSVLengthIndicated.cpp
C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/HeaderBuffer.cpp
C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/IndexFile.cpp

Functions

- void convertCSVToLengthIndicated (const std::string &csvFile, const std::string &outputFile)

  Converts a CSV file to a length-indicated format.
- std::vector< std::vector< std::string > > readLengthIndicatedRecord (const std::string &filename)

  Reads a length-indicated record from a file stream.

4.7.1 Detailed Description

Header file for functions to handle length-indicated file conversion and reading.

This header file declares functions for converting a CSV file into a length-indicated format and for reading records from a length-indicated file. The length-indicated format prefixes each record with the byte length of the record, facilitating variable-length data handling.

The length-indicated format is a custom structure where each record's length is stored before the actual data, allowing for efficient parsing of variable-length records. Functions included:

- `convertCSVToLengthIndicated()`: Converts CSV data to a length-indicated format.
- `readLengthIndicatedRecord()`: Reads a record from a length-indicated file.

Author

Thomas Hoerger

Date

October 18 2024

Definition in file CSVLengthIndicated.h.

4.7.2 Function Documentation

4.7.2.1 convertCSVToLengthIndicated()

```
void convertCSVToLengthIndicated (
            const std::string & csvFileName,
            const std::string & outputFileName)
```

Converts a CSV file to a length-indicated format.

Reads a CSV file, processes each record to prefix each field with its length, and writes the result to an output file in the length-indicated format.

Parameters

csvFile	The name of the input CSV file to be converted.
outputFile	The name of the output file where the length-indicated format data will be saved.

Note

The header row of the CSV is written without length indications, while all data rows have each field prefixed by a two-digit length indicator.

Reads each record from a CSV file and writes it to an output file, where each field in the record is prefixed by its length as a two-digit integer. The header row is written without length indicators.

Parameters

csvFileName	The name of the CSV file to be converted.
outputFileName	The name of the output file where length-indicated records will be stored.

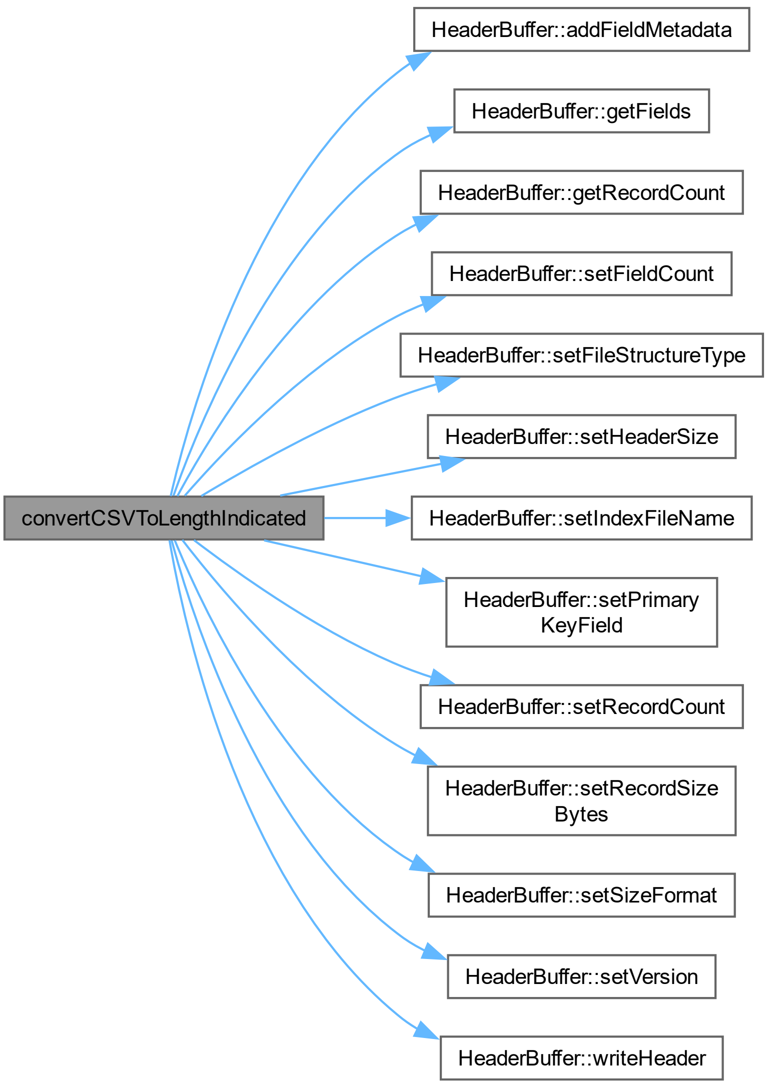
Note

Each field's length is formatted as a two-digit number, padded with zeroes if necessary.

If a field exceeds 99 characters, it will be truncated to fit within the two-digit length limit.

Definition at line 44 of file CSVLengthIndicated.cpp.

Here is the call graph for this function:


HeaderBuffer::addFieldMetadata
HeaderBuffer::getFields
HeaderBuffer::getRecordCount
HeaderBuffer::setFieldCount
HeaderBuffer::setFileStructureType
HeaderBuffer::setHeaderSize
convertCSVToLengthIndicated
HeaderBuffer::setIndexFileName
HeaderBuffer::setPrimary KeyField
HeaderBuffer::setRecordCount
HeaderBuffer::setRecordSize Bytes
HeaderBuffer::setSizeFormat
HeaderBuffer::setVersion
HeaderBuffer::writeHeader

4.7.2.2 readLengthIndicatedRecord()

```
std::vector< std::vector< std::string > > readLengthIndicatedRecord (
            const std::string & filename)
```

Reads a length-indicated record from a file stream.

Reads a single record from the provided length-indicated file stream. Each record is parsed by reading the specified length prefix before each field. The function returns the record data as a vector of strings, with each string representing a field in the record.

Parameters

fileStream	The input file stream from which to read the length-indicated record.

Returns

A vector of vectors of strings, where each inner vector represents a record read from the file.

Note

This function assumes each field is prefixed by its length as a two-digit integer.

Warning

The file stream should be opened in binary mode for correct reading.

Reads a length-indicated record from a file stream.

Reads each row of a CSV file into a vector of strings, where each inner vector represents a row in the CSV. The header row is ignored, and only data rows are returned.

Parameters

filename	The name of the CSV file to be read.

Returns

A vector of vectors, where each inner vector represents a row of fields.

Note

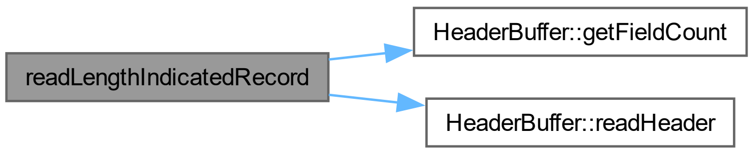
This function assumes a simple CSV structure with fields separated by commas.

Warning

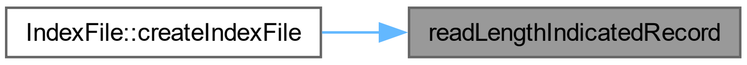
An error message is displayed if the file cannot be opened.

Definition at line 182 of file CSVLengthIndicated.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.8 CSVLengthIndicated.h

Go to the documentation of this file.

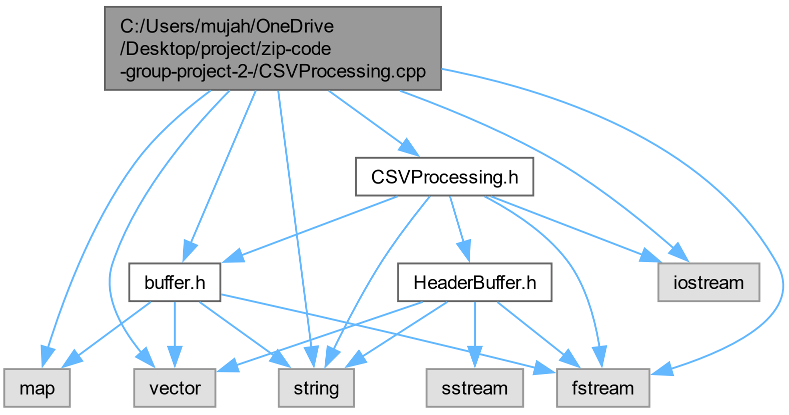
```

#ifndef CSV_LENGTH_INDICATED_H
#define CSV_LENGTH_INDICATED_H

#include <string>
#include <fstream>
#include <vector>

void convertCSVToLengthIndicated(const std::string &csvFile, const std::string &outputFile);

std::vector<std::vector<std::string>> readLengthIndicatedRecord( const std::string& filename );

#endif // CSV_LENGTH_INDICATED_H
```

4.9 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/CSVProcessing.cpp File Reference

```
#include "buffer.h"
#include "CSVProcessing.h"
#include <iostream>
#include <fstream>
```

```
#include <string>
#include <map>
#include <vector>
```

Include dependency graph for CSVProcessing.cpp:



4.10 CSVProcessing.cpp

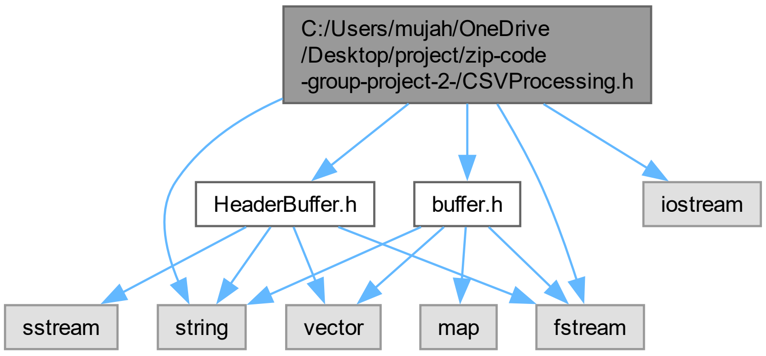
Go to the documentation of this file.

```
#include "buffer.h"
#include "CSVProcessing.h"
#include <iostream>
#include <fstream>
#include <string>
#include <map>
#include <vector>
//using namespace std;


// void CSVProcessing::printZipCodeRecord( const ZipCodeRecord& record ) {
//     std::cout « "Zip Code: " « record.zip_code
//         « ", State ID: " « record.state_id
//         « ", Latitude: " « record.latitude
//         « ", Longitude: " « record.longitude « std::endl;
// }
std::map<string, std::vector<ZipCodeRecord» CSVProcessing::sortBuffer() {
    float eastMost, westMost, northMost, southMost;
    Buffer CSVBuffer;
    CSVBuffer.read_csv( );
    std::map<string, std::vector<ZipCodeRecord» state_zip_map = CSVBuffer.get_state_zip_codes();
    std::map<string, std::vector<ZipCodeRecord» sorted_directions;
    for ( auto& state : state_zip_map ) {
        const std::string& stateID = state.first;
        const std::vector<ZipCodeRecord>& stateInfo = state.second;
        // intial loading of directions
        ZipCodeRecord easternmost = stateInfo[ 0 ];
        ZipCodeRecord westernmost = stateInfo[ 0 ];
        ZipCodeRecord northernmost = stateInfo[ 0 ];
        ZipCodeRecord southernmost = stateInfo[ 0 ];
        // checks if the current records zip is one of the maxed directions
        for ( const auto& record : stateInfo ) {
            if ( record.longitude < easternmost.longitude ) {
                easternmost = record;
            }
            if ( record.longitude > westernmost.longitude ) {
                westernmost = record;
            }
            if ( record.latitude > northernmost.latitude ) {
                northernmost = record;
            }
            if ( record.latitude < southernmost.latitude ) {
                southernmost = record;
            }
        }
        sorted_directions[ stateID ] = { easternmost, westernmost, northernmost, southernmost };
        // std::cout « "State: " « stateID « std::endl;
        // std::cout « "  Easternmost: ";
        // printZipCodeRecord( easternmost );
        // std::cout « "  Westernmost: ";
        // printZipCodeRecord( westernmost );
        // std::cout « "  Northernmost: ";
        // printZipCodeRecord( northernmost );
        // std::cout « "  Southernmost: ";
        // printZipCodeRecord( southernmost );
        // std::cout « std::endl;  // Add an extra line for readability
    }
    // sorted_directions looks like this
    // [stateID] : {
    //     { east most zip, stateID, directions },
    //     { west most zip, stateID, directions },
    //     { northern most zip, stateID, directions },
    //     { southern most zip, stateID, directions }
    // }

    return sorted_directions;
}

void CSVProcessing::addHeader(std::string& file_name) {
    std::ofstream file(file_name);
    if (file.is_open()) {
        file « "State,Easternmost,Westernmost,Northernmost,Southernmost\n";
        file.close();
        std::cout « "Header added successfully to " « file_name « std::endl;
    } else {
        std::cerr « "Unable to open file: " « file_name « std::endl;
    }
}
bool CSVProcessing::csvOutput(std::string& file_name) {
    std::map<std::string, std::vector<ZipCodeRecord» sorted_data = sortBuffer();
    std::ofstream file(file_name, std::ios::app);  // Open in append mode

    if (!file.is_open()) {
        std::cerr « "Unable to open file: " « file_name « std::endl;
        return false;
    }
    for (const auto& [state, records] : sorted_data) {
        if (records.size() == 4) {  // Ensure we have all 4 directional records
            file « state « ","
                 « records[0].zip_code « ","  // Easternmost
                 « records[1].zip_code « ","  // Westernmost
                 « records[2].zip_code « ","  // Northernmost
                 « records[3].zip_code « "\n";  // Southernmost
        }
    }
    file.close();
    std::cout « "Data successfully written to " « file_name « std::endl;
    return true;
}
```

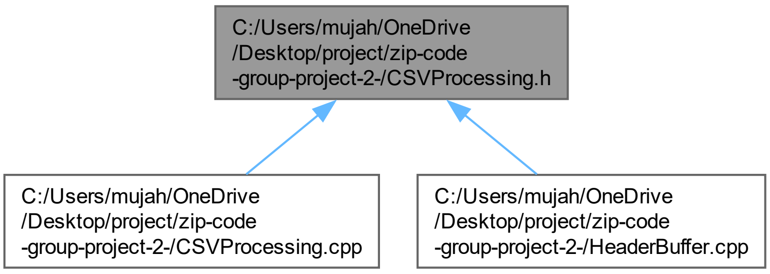
4.11 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/CSVProcessing.h File Reference

```
#include "buffer.h"
#include <iostream>
#include <fstream>
#include <string>
#include "HeaderBuffer.h"
```

Include dependency graph for CSVProcessing.h:



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


C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/CSVProcessing.h
C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/CSVProcessing.cpp
C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/HeaderBuffer.cpp

Classes

- class CSVProcessing

4.12 CSVProcessing.h

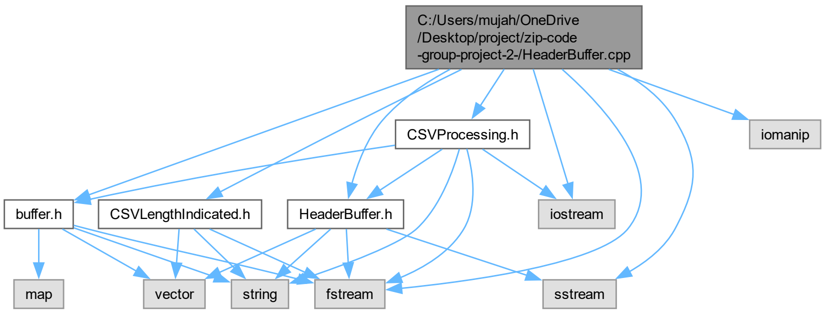
Go to the documentation of this file.

```
#ifndef CSVProcessing_H
#define CSVProcessing_H

#include "buffer.h"
#include <iostream>
#include <fstream>
#include <string>
#include "HeaderBuffer.h"

using namespace std;

class CSVProcessing {
public:
    map<string, vector<ZipCodeRecord» sortBuffer(); // sort by state with the hashmap but how once it
is sorted we can do the direction farthest zip
    // we could also set up a const variable that will have the state ids based on their index/hasmap
key and with that we can instantlly find where the zip should go
    //void printZipCodeRecord( const ZipCodeRecord& record ); for testing purposes
    void addHeader( string& file_name ); // state id, Easternmost (least longitude), Westernmost,
Northernmost (greatest latitude), and Southernmost Zip Code
    bool csvOutput( string& file_name ); // fill from the sortered buffer? either output as we go from
the buffer or create an array or vector to put all the sorting and then output to the csv
};

#endif
```

4.13 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/HeaderBuffer.cpp File Reference

```
#include "CSVProcessing.h"
#include "buffer.h"
#include "CSVLengthIndicated.h"
#include "HeaderBuffer.h"
#include <iostream>
#include <fstream>
#include <sstream>
#include <iomanip>
```

Include dependency graph for HeaderBuffer.cpp:



4.14 HeaderBuffer.cpp

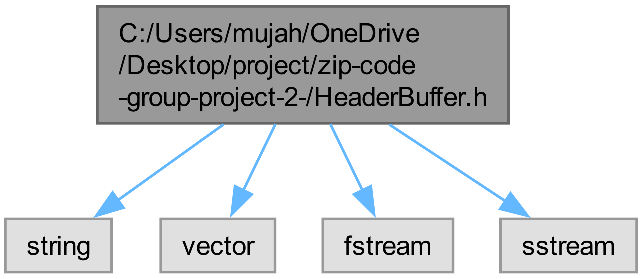
Go to the documentation of this file.

```
#include "CSVProcessing.h"
#include "buffer.h"
#include "CSVLengthIndicated.h"
#include "HeaderBuffer.h"
#include <iostream>
#include <fstream>
#include <sstream>
#include <iomanip>

HeaderBuffer::HeaderBuffer()
    : headerRecordSize(0)
    , recordSizeBytes(0)
    , recordCount(0)
    , fieldCount(0)
    , primaryKeyField(0) {
}

bool HeaderBuffer::writeHeader(const std::string& filename) {
    std::ofstream file(filename);
    if (!file.is_open()) {
        std::cerr « "Error: Unable to open file for writing: " « filename « std::endl;
        return false;
    }

    auto writeField = [&file](const std::string& value) {
        std::string lengthStr = std::to_string(value.length());
        if (lengthStr.length() < 2) lengthStr = "0" + lengthStr;
        file « lengthStr « value;
    };

    // Write main header fields
    writeField(fileStructureType); file « ",";
    writeField(version); file « ",";
    writeField(std::to_string(headerRecordSize)); file « ",";
    writeField(std::to_string(recordSizeBytes)); file « ",";
    writeField(sizeFormatType); file « ",";
    writeField(indexFileName); file « ",";
    writeField(std::to_string(recordCount)); file « ",";
    writeField(std::to_string(fieldCount)); file « ",";
    writeField(std::to_string(primaryKeyField));
    file « "\n";

    // Write field metadata
    for (const auto& field : fields) {
        writeField(field.name); file « ",";
        writeField(field.typeSchema); file « "\n";
    }

    file.close();
    return true;
}

bool HeaderBuffer::readHeader(const std::string& filename) {
    std::ifstream file(filename);
    if (!file.is_open()) {
        std::cerr « "Error: Unable to open file for reading: " « filename « std::endl;
        return false;
    }

    std::string line;
    if (std::getline(file, line)) {
        std::stringstream ss(line);

        // Helper function to read length-indicated field
        auto readField = [](std::stringstream& ss) -> std::string {
            std::string lenStr;
            lenStr.resize(2);
            ss.read(&lenStr[0], 2);

            if (!std::isdigit(lenStr[0]) || !std::isdigit(lenStr[1])) {
                throw std::runtime_error("Invalid length indicator");
            }

            int length = std::stoi(lenStr);
            std::string value;
            value.resize(length);
            ss.read(&value[0], length);

            if (ss.peek() == ',') ss.ignore();
            return value;
        };

        try {
            fileStructureType = readField(ss);
            version = readField(ss);
            headerRecordSize = std::stoi(readField(ss));
            recordSizeBytes = std::stoi(readField(ss));
            sizeFormatType = readField(ss);
            indexFileName = readField(ss);
            recordCount = std::stoi(readField(ss));
            fieldCount = std::stoi(readField(ss));
            primaryKeyField = std::stoi(readField(ss));

            fields.clear();
            for (int i = 0; i < fieldCount && std::getline(file, line); i++) {
                std::stringstream fieldSS(line);
                FieldMetadata metadata;
                metadata.name = readField(fieldSS);
                metadata.typeSchema = readField(fieldSS);
                fields.push_back(metadata);
            }
        } catch (const std::exception& e) {
            std::cerr « "Error parsing header: " « e.what() « std::endl;
            return false;
        }
    }

    file.close();
    return true;
}
```

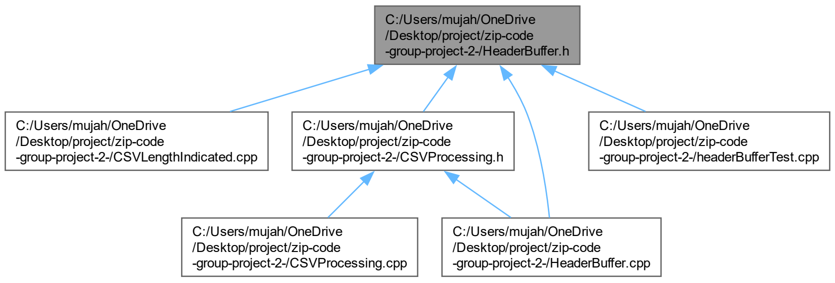
4.15 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/HeaderBuffer.h File Reference

```
#include <string>
#include <vector>
#include <fstream>
#include <sstream>
```

Include dependency graph for HeaderBuffer.h:



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


C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/HeaderBuffer.h
C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/CSVLengthIndicated.cpp
C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/CSVProcessing.h
C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/headerBufferTest.cpp
C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/CSVProcessing.cpp
C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/HeaderBuffer.cpp

Classes

- struct FieldMetadata

  Structure to hold field metadata information.
- class HeaderBuffer

  Class for handling the data file header record.

4.16 HeaderBuffer.h

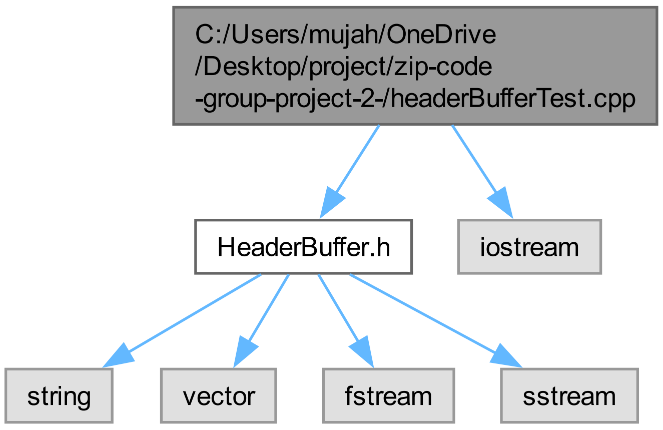
Go to the documentation of this file.

```
#ifndef HEADER_BUFFER_H
#define HEADER_BUFFER_H

#include <string>
#include <vector>
#include <fstream>
#include <sstream>

struct FieldMetadata {
    std::string name;       // Field name/ID
    std::string typeSchema; // Format to read/write
};

class HeaderBuffer {
private:
    std::string fileStructureType;  // Type of file structure
    std::string version;            // Version of file structure
    int headerRecordSize;           // Size of header record in bytes
    int recordSizeBytes;            // Bytes for each record size integer
    std::string sizeFormatType;     // ASCII or binary
    std::string indexFileName;      // Name of primary key index file
    int recordCount;                // Number of records in file
    int fieldCount;                 // Number of fields per record
    int primaryKeyField;            // Ordinal number of primary key field
    std::vector<FieldMetadata> fields; // Metadata for each field

public:
    // Constructor
    HeaderBuffer();

    // File operations
    bool writeHeader(const std::string& filename);
    bool readHeader(const std::string& filename);

    // Setters
    void setFileStructureType(const std::string& type) { fileStructureType = type; }
    void setVersion(const std::string& ver) { version = ver; }
    void setHeaderSize(int size) { headerRecordSize = size; }
    void setRecordSizeBytes(int bytes) { recordSizeBytes = bytes; }
    void setSizeFormat(const std::string& format) { sizeFormatType = format; }
    void setIndexFileName(const std::string& name) { indexFileName = name; }
    void setRecordCount(int count) { recordCount = count; }
    void setFieldCount(int count) { fieldCount = count; }
    void setPrimaryKeyField(int field) { primaryKeyField = field; }
    void addFieldMetadata(const std::string& name, const std::string& schema) {
        fields.push_back({name, schema});
    }

    // Getters
    std::string getFileStructureType() const { return fileStructureType; }
    std::string getVersion() const { return version; }
    int getHeaderSize() const { return headerRecordSize; }
    int getRecordSizeBytes() const { return recordSizeBytes; }
    std::string getSizeFormat() const { return sizeFormatType; }
    std::string getIndexFileName() const { return indexFileName; }
    int getRecordCount() const { return recordCount; }
    int getFieldCount() const { return fieldCount; }
    int getPrimaryKeyField() const { return primaryKeyField; }
    const std::vector<FieldMetadata>& getFields() const { return fields; }
};

#endif // HEADER_BUFFER_H
```

4.17 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/headerBufferTest.cpp File Reference

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

Include dependency graph for headerBufferTest.cpp:



Functions

- int main ()

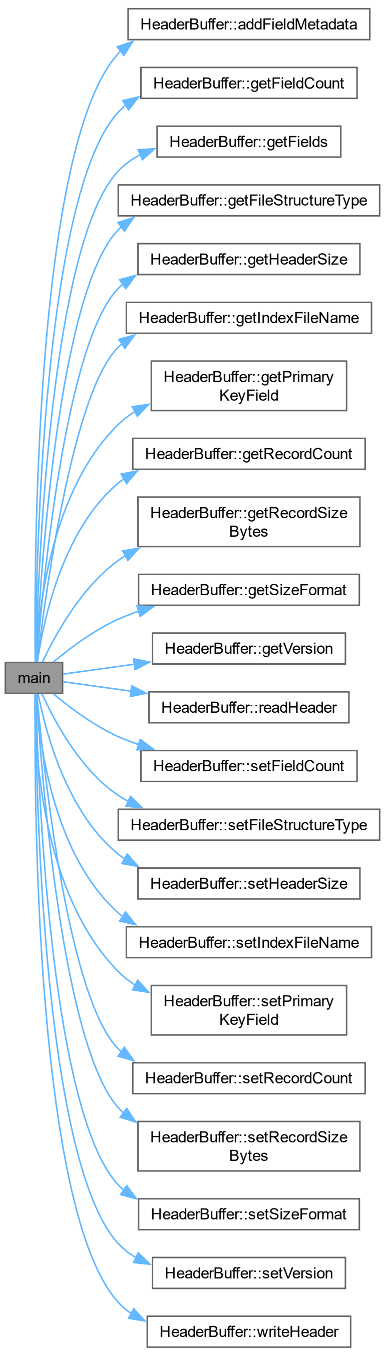
4.17.1 Function Documentation

4.17.1.1 main()

```
int main ()
```

Definition at line 4 of file headerBufferTest.cpp.

Here is the call graph for this function:


main
HeaderBuffer::addFieldMetadata
HeaderBuffer::getFieldCount
HeaderBuffer::getFields
HeaderBuffer::getFileStructureType
HeaderBuffer::getHeaderSize
HeaderBuffer::getIndexFileName
HeaderBuffer::getPrimary KeyField
HeaderBuffer::getRecordCount
HeaderBuffer::getRecordSize Bytes
HeaderBuffer::getSizeFormat
HeaderBuffer::getVersion
HeaderBuffer::readHeader
HeaderBuffer::setFieldCount
HeaderBuffer::setFileStructureType
HeaderBuffer::setHeaderSize
HeaderBuffer::setIndexFileName
HeaderBuffer::setPrimary KeyField
HeaderBuffer::setRecordCount
HeaderBuffer::setRecordSize Bytes
HeaderBuffer::setSizeFormat
HeaderBuffer::setVersion
HeaderBuffer::writeHeader

4.18 headerBufferTest.cpp

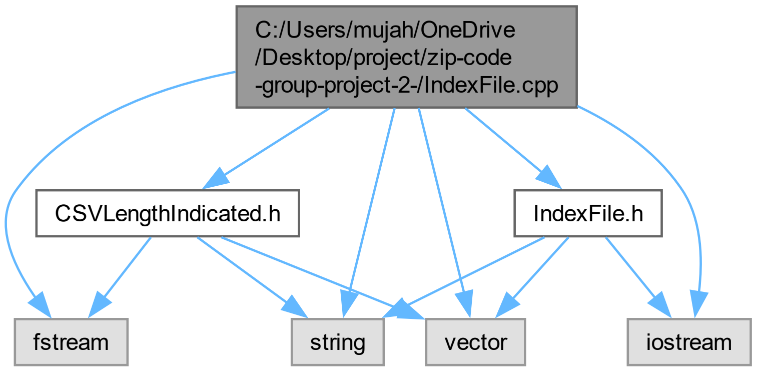
Go to the documentation of this file.

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

int main() {
    HeaderBuffer header;

    // Set header fields
    header.setFileStructureType("CSV");
    header.setVersion("1.0");
    header.setHeaderSize(256);
    header.setRecordSizeBytes(4);
    header.setSizeFormat("ASCII");
    header.setIndexFileName("headerBufferTest.csv");
    header.setRecordCount(1000);
    header.setFieldCount(6);
    header.setPrimaryKeyField(0);

    // Add field metadata
    header.addFieldMetadata("test", "string");

    // Write header to file
    if (header.writeHeader("headerBufferTest.csv")) {
        std::cout « "Header written to zipcode_data.csv" « std::endl;
    } else {
        std::cerr « "Failed to write header" « std::endl;
    }

    // Read header from file
    HeaderBuffer readHeader;
    if (readHeader.readHeader("headerBufferTest.csv")) {
        std::cout « "Header read successfully:" « std::endl;
        std::cout « "File structure type: " « readHeader.getFileStructureType() « std::endl;
        std::cout « "Version: " « readHeader.getVersion() « std::endl;
        std::cout « "Header size: " « readHeader.getHeaderSize() « std::endl;
        std::cout « "Record size bytes: " « readHeader.getRecordSizeBytes() « std::endl;
        std::cout « "Size format: " « readHeader.getSizeFormat() « std::endl;
        std::cout « "Index file name: " « readHeader.getIndexFileName() « std::endl;
        std::cout « "Record count: " « readHeader.getRecordCount() « std::endl;
        std::cout « "Field count: " « readHeader.getFieldCount() « std::endl;
        std::cout « "Primary key field: " « readHeader.getPrimaryKeyField() « std::endl;

        const auto& fields = readHeader.getFields();
        std::cout « "Field metadata:" « std::endl;
        for (const auto& field : fields) {
            std::cout « "  Name: " « field.name « ", Type: " « field.typeSchema « std::endl;
        }
    } else {
        std::cerr « "Failed to read header" « std::endl;
    }

    return 0;
}
```

4.19 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/IndexFile.cpp File Reference

```
#include "CSVLengthIndicated.h"
#include "IndexFile.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
```

Include dependency graph for IndexFile.cpp:



4.20 IndexFile.cpp

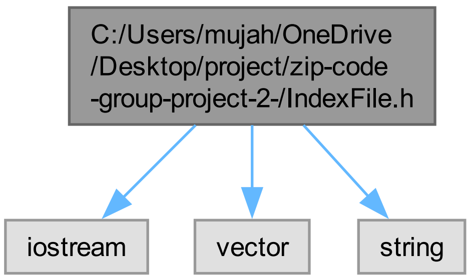
Go to the documentation of this file.

```
#include "CSVLengthIndicated.h"
#include "IndexFile.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
// zipcode, length offeset
// zipcode, length offeset

bool IndexFile::createIndexFile( const std::string& csvFileName, const std::string& outputFileName ) {
    std::ofstream outputFile( outputFileName );
    if ( !outputFile.is_open() ) {
        std::cerr « "Failed to open output file." « std::endl;
        return false;
    }

    int cumulativeOffset = 0;  // Initialize cumulative offset
    std::vector<std::vector<std::string» records = readLengthIndicatedRecord( csvFileName );

    // outputFile « records[ 0 ][ 0 ].substr( 2 ) « " " « 0 « std::endl;
    int rowOffset = 0;  // Initialize offset for the current row

    // Loop through all rows
    for ( const auto& row : records ) {
        if ( row.empty() ) continue;  // Skip empty rows
        cumulativeOffset += rowOffset;
        // Extract and correct the zip code (first field of the row)
        std::string correctedZip = row[ 0 ].substr( 2 );

        // Write the corrected zip code and the cumulative offset to the output file
        outputFile « correctedZip « " " « cumulativeOffset « std::endl;
        rowOffset = 0;

        // Nested loop: Process each field within the current row
        for ( const auto& field : row ) {
            if ( field.size() >= 2 ) {  // Ensure the field has at least two characters
                std::string offsetInString = field.substr( 0, 2 );  // Get the first two characters
                int fieldOffset = std::stoi( offsetInString );  // Convert to integer

                rowOffset += fieldOffset + 2;  // Add to the row's total offset
            }
        }
        rowOffset += 5;

        // Add the row's total offset to the cumulative offset
    }
    outputFile.close();  // Close the output file
    return true;
}
```

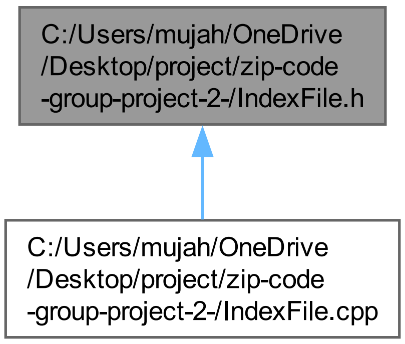
4.21 C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/IndexFile.h File Reference

```
#include <iostream>
#include <vector>
#include <string>
```

Include dependency graph for IndexFile.h:



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


C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/IndexFile.h
C:/Users/mujah/OneDrive
/Desktop/project/zip-code
-group-project-2-/IndexFile.cpp

Classes

- class IndexFile

4.22 IndexFile.h

Go to the documentation of this file.

```
#ifndef INDEX_FILE_H
#define INDEX_FILE_H

#include <iostream>
#include <vector>
#include <string>

class IndexFile {
    public:
        bool createIndexFile( const std::string& csvFile, const std::string& outputFile );
};

#endif
```

Index

addFieldMetadata
    HeaderBuffer, 14
addHeader
    CSVProcessing, 9

Buffer, 5
    get_state_zip_codes, 6
    parse_csv_line, 6
    read_csv, 7
    readLengthIndicatedRecord, 7
    records, 8

C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/buffer.cpp, 27, 28
C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/buffer.h, 30, 31
C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/CSVLengthIndicated.cpp, 31, 35
C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/CSVLengthIndicated.h, 38, 42
C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/CSVProcessing.cpp, 42, 43
C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/CSVProcessing.h, 44, 45
C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/HeaderBuffer.cpp, 46
C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/HeaderBuffer.h, 48, 49
C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/headerBufferTest.cpp, 49, 51
C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/IndexFile.cpp, 52, 53
C:/Users/mujah/OneDrive/Desktop/project/zip-code-group-project-2-/IndexFile.h, 54
city
    ZipCodeRecord, 26
convertCSVToLengthIndicated
    CSVLengthIndicated.cpp, 33
    CSVLengthIndicated.h, 39
createIndexFile
    IndexFile, 25
CSVLengthIndicated.cpp
    convertCSVToLengthIndicated, 33
    readLengthIndicatedRecord, 34
CSVLengthIndicated.h
    convertCSVToLengthIndicated, 39
    readLengthIndicatedRecord, 40
csvOutput
    CSVProcessing, 9
CSVProcessing, 8
    addHeader, 9
    csvOutput, 9
    sortBuffer, 10

fieldCount
    HeaderBuffer, 23
FieldMetadata, 11
    name, 11
    typeSchema, 11
fields
    HeaderBuffer, 23
fileStructureType
    HeaderBuffer, 23

get_state_zip_codes
    Buffer, 6
getFieldCount
    HeaderBuffer, 14
getFields
    HeaderBuffer, 14
getFileStructureType
    HeaderBuffer, 15
getHeaderSize
    HeaderBuffer, 15
getIndexFileName
    HeaderBuffer, 15
getPrimaryKeyField
    HeaderBuffer, 16
getRecordCount
    HeaderBuffer, 16
getRecordSizeBytes
    HeaderBuffer, 16
getSizeFormat
    HeaderBuffer, 17
getVersion
    HeaderBuffer, 17

HeaderBuffer, 12
    addFieldMetadata, 14
    fieldCount, 23
    fields, 23
    fileStructureType, 23
    getFieldCount, 14
    getFields, 14
    getFileStructureType, 15
    getHeaderSize, 15
    getIndexFileName, 15

getPrimaryKeyField, 16
getRecordCount, 16
getRecordSizeBytes, 16
getSizeFormat, 17
getVersion, 17
HeaderBuffer, 13
headerRecordSize, 23
indexFileName, 23
primaryKeyField, 23
readHeader, 17
recordCount, 23
recordSizeBytes, 23
setFieldCount, 18
setFileStructureType, 18
setHeaderSize, 19
setIndexFileName, 19
setPrimaryKeyField, 20
setRecordCount, 20
setRecordSizeBytes, 20
setSizeFormat, 21
setVersion, 21
sizeFormatType, 24
version, 24
writeHeader, 22

headerBufferTest.cpp
main, 50

headerRecordSize
HeaderBuffer, 23

IndexFile, 24
createIndexFile, 25

indexFileName
HeaderBuffer, 23

latitude
ZipCodeRecord, 26

longitude
ZipCodeRecord, 26

main
headerBufferTest.cpp, 50

name
FieldMetadata, 11

parse_csv_line
Buffer, 6

primaryKeyField
HeaderBuffer, 23

read_csv
Buffer, 7

readHeader
HeaderBuffer, 17

readLengthIndicatedRecord
Buffer, 7
CSVLengthIndicated.cpp, 34
CSVLengthIndicated.h, 40

recordCount
HeaderBuffer, 23

records
Buffer, 8

recordSizeBytes
HeaderBuffer, 23

setFieldCount
HeaderBuffer, 18

setFileStructureType
HeaderBuffer, 18

setHeaderSize
HeaderBuffer, 19

setIndexFileName
HeaderBuffer, 19

setPrimaryKeyField
HeaderBuffer, 20

setRecordCount
HeaderBuffer, 20

setRecordSizeBytes
HeaderBuffer, 20

setSizeFormat
HeaderBuffer, 21

setVersion
HeaderBuffer, 21

sizeFormatType
HeaderBuffer, 24

sortBuffer
CSVProcessing, 10

state_id
ZipCodeRecord, 26

typeSchema
FieldMetadata, 11

version
HeaderBuffer, 24

writeHeader
HeaderBuffer, 22

zip_code
ZipCodeRecord, 26

ZipCodeRecord, 25
city, 26
latitude, 26
longitude, 26
state_id, 26
zip_code, 26