Processing CSV Files

Fabian, Chakong, Shishir,Abdirahman,Hamad and Roshan

Generated by Doxygen 1.9.8

1 Class Index 1

1.1 Class List 1

2 File Index 3

2.1 File List 3

3 Class Documentation 5

3.1 CommandLineReader Class Reference 5

3.1.1 Detailed Description 6

3.1.2 Constructor & Destructor Documentation 6

3.1.2.1 CommandLineReader() 6

3.1.3 Member Function Documentation 6

3.1.3.1 Main() 6

3.1.3.2 ParseCommandLine() 7

3.1.4 Member Data Documentation 8

3.1.4.1 running 8

3.2 CSVReader Class Reference 9

3.2.1 Detailed Description 10

3.2.2 Constructor & Destructor Documentation 10

3.2.2.1 CSVReader() 10

3.2.2.2 ∼CSVReader() 11

3.2.3 Member Function Documentation 11

3.2.3.1 BuildDataFile() 11

3.2.3.2 buildFileStructure() 11

3.2.3.3 close() 12

3.2.3.4 ConvertToLength() 12

3.2.3.5 GenerateHeaderRecord() 12

3.2.3.6 GetHeaders() 12

3.2.3.7 isOpen() 13

3.2.3.8 ParseLine() 14

3.2.3.9 ReadFromFile() 15

3.2.3.10 WriteToFile() 15

3.2.4 Member Data Documentation 15

3.2.4.1 Headers 15

3.2.4.2 ZipCSV 16

3.3 HeaderRecord Class Reference 16

3.3.1 Detailed Description 18

3.3.2 Constructor & Destructor Documentation 18

3.3.2.1 HeaderRecord() 18

3.3.3 Member Function Documentation 19

3.3.3.1 getFieldsPerRecord() 19

3.3.3.2 getFileName() 19

3.3.3.3 getHeaderSize() 19

3.3.3.4 getPrimaryKeyIndexFileName() 19
3.3.3.5 getPrimaryKeyOrdinality() 19
3.3.3.6 getRecordCount() 19
3.3.3.7 getRecordSizeBytes() 19
3.3.3.8 getSizeFormatType() 20
3.3.3.9 getVersion() 20
3.3.3.10 setFieldsPerRecord() 20
3.3.3.11 setFileName() 20
3.3.3.12 setHeaderSize() 20
3.3.3.13 setPrimaryKeyIndexFileName() 20
3.3.3.14 setPrimaryKeyOrdinality() 21
3.3.3.15 setRecordCount() 21
3.3.3.16 setRecordSizeBytes() 21
3.3.3.17 setSizeFormatType() 21
3.3.3.18 setVersion() 21
3.3.4 Member Data Documentation 22
3.3.4.1 fieldNames 22
3.3.4.2 fieldsPerRecord 22
3.3.4.3 fileName 22
3.3.4.4 headerSize 22
3.3.4.5 primaryKeyIndexFileName 22
3.3.4.6 primaryKeyOrdinality 22
3.3.4.7 recordCount 22
3.3.4.8 recordSizeBytes 23
3.3.4.9 sizeFormatType 23
3.3.4.10 version 23
3.4 HeaderRecordBuffer Class Reference 23
3.4.1 Detailed Description 25
3.4.2 Constructor & Destructor Documentation 25
3.4.2.1 HeaderRecordBuffer() 25
3.4.2.2 ∼HeaderRecordBuffer() 26
3.4.3 Member Function Documentation 26
3.4.3.1 GetHeaderRecord() 26
3.4.3.2 lengthDecoder() 27
3.4.3.3 parser() 28
3.4.3.4 ReadHeaderRecord() 29
3.4.3.5 SetHeaderRecord() 30
3.4.3.6 WriteHeaderRecord() 31
3.4.4 Member Data Documentation 31
3.4.4.1 headerRecord 31
3.5 PrimaryKeyIndex Class Reference 32
3.5.1 Detailed Description 33

3.5.2 Constructor & Destructor Documentation 33
3.5.2.1 PrimaryKeyIndex() 33
3.5.3 Member Function Documentation 33
3.5.3.1 BuildIndex() 33
3.5.3.2 ReadIndex() 34
3.5.3.3 SearchIndex() 35
3.5.3.4 UnpackRecord() 36
3.5.3.5 WriteIndex() 36
3.6 Record Struct Reference 37
3.6.1 Detailed Description 38
3.6.2 Member Data Documentation 38
3.6.2.1 recordData 38
3.6.2.2 recordLength 38
3.7 Row Struct Reference 39
3.7.1 Detailed Description 40
3.7.2 Member Data Documentation 40
3.7.2.1 county 40
3.7.2.2 latitude 40
3.7.2.3 longitude 40
3.7.2.4 name 40
3.7.2.5 state 40
3.7.2.6 zip 40

4 File Documentation **41**
4.1 CommandLineReader.cpp File Reference 41
4.1.1 Detailed Description 41
4.2 CommandLineReader.cpp 42
4.3 CommandLineReader.h File Reference 43
4.3.1 Detailed Description 44
4.4 CommandLineReader.h 45
4.5 CSVReader.cpp File Reference 45
4.5.1 Detailed Description 45
4.6 CSVReader.cpp 47
4.7 CSVReader.h File Reference 48
4.7.1 Detailed Description 49
4.8 CSVReader.h 50
4.9 HeaderRecord.cpp File Reference 51
4.10 HeaderRecord.cpp 51
4.11 HeaderRecord.h File Reference 52
4.12 HeaderRecord.h 53
4.13 HeaderRecordBuffer.cpp File Reference 54
4.13.1 Detailed Description 54

4.14 HeaderRecordBuffer.cpp 55
4.15 HeaderRecordBuffer.h File Reference 57
4.15.1 Detailed Description 58
4.16 HeaderRecordBuffer.h 59
4.17 main.cpp File Reference 59
4.17.1 Detailed Description 60
4.17.2 Function Documentation 60
4.17.2.1 analyzeCSV() 60
4.17.2.2 main() 61
4.18 main.cpp 62
4.19 PrimaryKeyIndex.cpp File Reference 63
4.19.1 Detailed Description 63
4.20 PrimaryKeyIndex.cpp 63
4.21 PrimaryKeyIndex.h File Reference 65
4.21.1 Detailed Description 66
4.22 PrimaryKeyIndex.h 66

Index **67**

Chapter 1

Class Index

1.1 Class List

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

CommandLineReader 5

CSVReader 9

HeaderRecord 16

HeaderRecordBuffer 23

PrimaryKeyIndex

Represents the Primary Key Index functionality. This class provides methods for building, reading, writing, searching, and unpacking a primary key index 32

Record 37

Row

Represents a row of data in the CSV file. This struct stores information for a single row of data in the CSV file, including the ZIP code, name, state, county, latitude, and longitude 39

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

CommandLineReader.cpp
Member function definitions for class CommandLineReader 41
CommandLineReader.h
Declarations for class CommandLineReader 43
CSVReader.cpp
Member function definitions for class CSVReader 45
CSVReader.h
Declarations for class CSVReader 48
HeaderRecord.cpp 51
HeaderRecord.h 52
HeaderRecordBuffer.cpp
Definitions for class HeaderRecordBuffer 54
HeaderRecordBuffer.h
Declarations for class HeaderRecordBuffer 57
main.cpp
This program reads a CSV file containing postal code data, calculates state statistics, and displays the easternmost, westernmost, northernmost, and southernmost locations for each state. It also makes a CommandLineReader instance to check for zipcodes and if location is present 59
PrimaryKeyIndex.cpp
Member function definitions for the PrimaryKeyIndex class 63
PrimaryKeyIndex.h
Declarations for class PrimaryKeyIndex 65

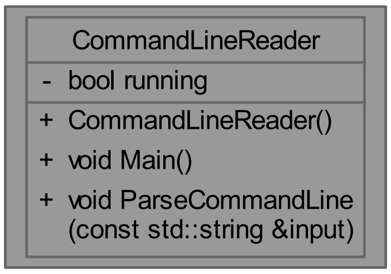
Chapter 3

Class Documentation

3.1 CommandLineReader Class Reference

```
#include <CommandLineReader.h>
```

Collaboration diagram for CommandLineReader:



Public Member Functions

- CommandLineReader ()

  Default constructor for CommandLineReader.
- void Main ()

  Starts the command line reader loop, processing inputs.
- void ParseCommandLine (const std::string &input)

  Parses the provided command line input string.

Private Attributes

- bool running

3.1.1 Detailed Description

Definition at line 23 of file CommandLineReader.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 CommandLineReader()

```
CommandLineReader::CommandLineReader ( )
```

Default constructor for CommandLineReader.

Default constructor that initializes the application state.

Precondition

None.

Postcondition

A CommandLineReader object is constructed with default settings.

Precondition

None.

Postcondition

The CommandLineReader object is constructed and the application is ready to run.

Definition at line 32 of file CommandLineReader.cpp.

3.1.3 Member Function Documentation

3.1.3.1 Main()

```
void CommandLineReader::Main ( )
```

Starts the command line reader loop, processing inputs.

Main loop that displays the menu and accepts commands from the user.

Precondition

None.

Postcondition

Command lines are processed until the reader is terminated.

Precondition

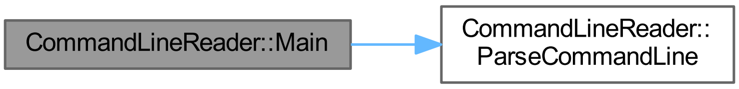
The CommandLineReader object is initialized.

Postcondition

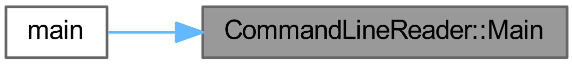
The user has interacted with the application, and possibly chosen to exit.

Definition at line 41 of file CommandLineReader.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.3.2 ParseCommandLine()

```
void CommandLineReader::ParseCommandLine (
             const std::string & input )
```

Parses the provided command line input string.

Parses and executes the command entered by the user.

Parameters

input	The command line input as a string.

Precondition

Valid input string is provided.

Postcondition

The input string is parsed and appropriate actions are taken based on the content.

Parameters

input	The user's input string.

Precondition

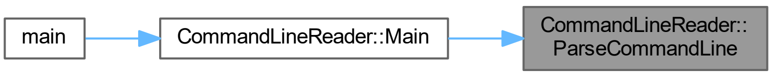
The application is running and waiting for user input.

Postcondition

The user's command is executed.

Definition at line 61 of file CommandLineReader.cpp.

Here is the caller graph for this function:



3.1.4 Member Data Documentation

3.1.4.1 running

```
bool CommandLineReader::running  [private]
```

Flag to check if the program is still running.

Definition at line 48 of file CommandLineReader.h.

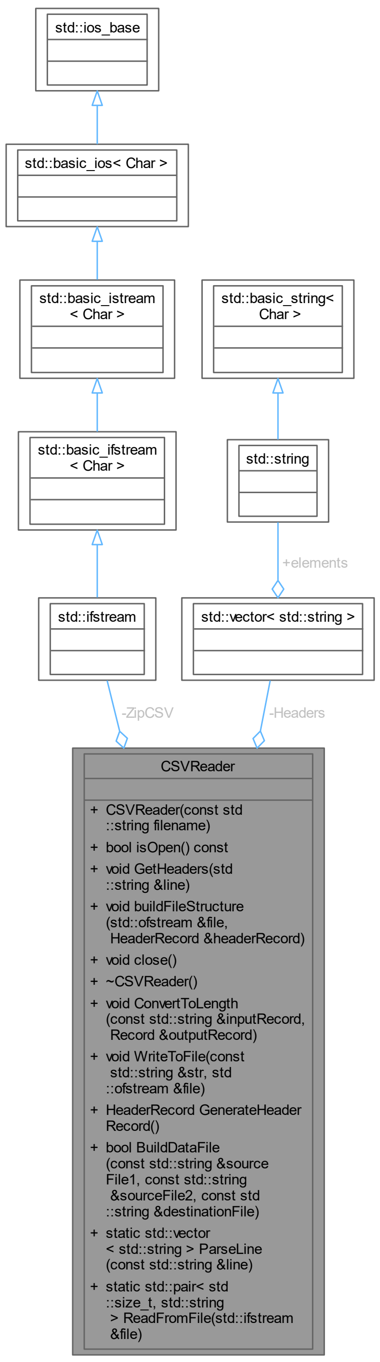
The documentation for this class was generated from the following files:

- CommandLineReader.h
- CommandLineReader.cpp

3.2 CSVReader Class Reference

```
#include <CSVReader.h>
```

Collaboration diagram for CSVReader:


std::ios_base
std::basic_ios< Char >
std::basic_istream < Char >
std::basic_string< Char >
std::basic_ifstream < Char >
std::string
+elements
std::ifstream
std::vector< std::string >
-ZipCSV
-Headers
CSVReader
+ CSVReader(const std ::string filename)
+ bool isOpen() const
+ void GetHeaders(std ::string &line)
+ void buildFileStructure (std::ofstream &file, HeaderRecord &headerRecord)
+ void close()
+ ~CSVReader()
+ void ConvertToLength (const std::string &inputRecord, Record &outputRecord)
+ void WriteToFile(const std::string &str, std ::ofstream &file)
+ HeaderRecord GenerateHeader Record()
+ bool BuildDataFile (const std::string &source File1, const std::string &sourceFile2, const std ::string &destinationFile)
+ static std::vector < std::string > ParseLine (const std::string &line)
+ static std::pair< std ::size_t, std::string > ReadFromFile(std::ifstream &file)

Public Member Functions

- CSVReader (const std::string filename)

Constructor that opens the CSV file specified by the 'filename' parameter.
- bool isOpen () const

  Checks if the CSV file is open.
- void GetHeaders (std::string &line)

  Parses and stores the header row of the CSV file.
- void buildFileStructure (std::ofstream &file, HeaderRecord &headerRecord)

  Reads and processes the entire CSV file.
- void close ()

  Closes the CSV file if it's open.
- ∼CSVReader ()
- void ConvertToLength (const std::string &inputRecord, Record &outputRecord)
- void WriteToFile (const std::string &str, std::ofstream &file)
- HeaderRecord GenerateHeaderRecord ()
- bool BuildDataFile (const std::string &sourceFile1, const std::string &sourceFile2, const std::string &destinationFile)

Static Public Member Functions

- static std::vector< std::string > ParseLine (const std::string &line)

  Parses a single data row of the CSV file into a Row object.
- static std::pair< std::size_t, std::string > ReadFromFile (std::ifstream &file)

Private Attributes

- std::ifstream ZipCSV
- std::vector< std::string > Headers

3.2.1 Detailed Description

Definition at line 56 of file CSVReader.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 CSVReader()

```
CSVReader::CSVReader (
            const std::string filename )
```

Constructor that opens the CSV file specified by the 'filename' parameter.

Parameters

filename	The name of the CSV file to open.

Precondition

None.

Postcondition

The CSVReader object is constructed, and the CSV file is opened for reading.

Definition at line 46 of file CSVReader.cpp.

3.2.2.2 ∼CSVReader()

```
CSVReader::∼CSVReader ( )
```

Definition at line 187 of file CSVReader.cpp.

3.2.3 Member Function Documentation

3.2.3.1 BuildDataFile()

```
bool CSVReader::BuildDataFile (
            const std::string & sourceFile1,
            const std::string & sourceFile2,
            const std::string & destinationFile )
```

Definition at line 155 of file CSVReader.cpp.

3.2.3.2 buildFileStructure()

```
void CSVReader::buildFileStructure (
            std::ofstream & file,
            HeaderRecord & headerRecord )
```

Reads and processes the entire CSV file.

Precondition

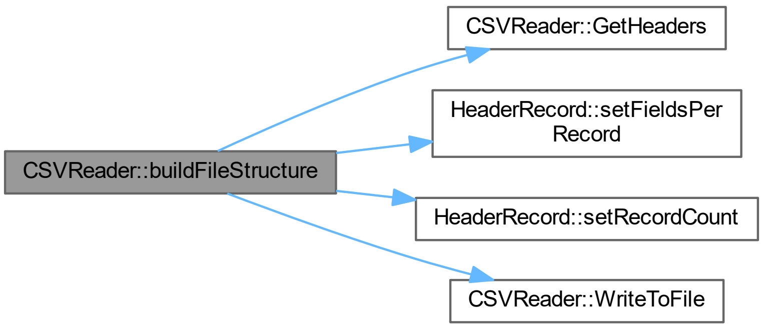
The CSV file is open for reading.

Postcondition

The CSV file is read, and data is parsed and stored in memory.

Definition at line 81 of file CSVReader.cpp.

Here is the call graph for this function:


CSVReader::GetHeaders
HeaderRecord::setFieldsPer Record
CSVReader::buildFileStructure
HeaderRecord::setRecordCount
CSVReader::WriteToFile

3.2.3.3 close()

```
void CSVReader::close ( )
```

Closes the CSV file if it's open.

Precondition

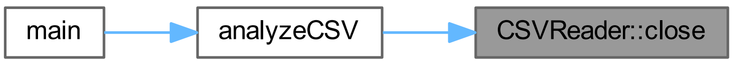
None.

Postcondition

The CSV file is closed if it was open.

Definition at line 181 of file CSVReader.cpp.

Here is the caller graph for this function:



3.2.3.4 ConvertToLength()

```
void CSVReader::ConvertToLength (
             const std::string & inputRecord,
             Record & outputRecord )
```

Definition at line 122 of file CSVReader.cpp.

3.2.3.5 GenerateHeaderRecord()

```
HeaderRecord CSVReader::GenerateHeaderRecord ( )
```

3.2.3.6 GetHeaders()

```
void CSVReader::GetHeaders (
             std::string & line )
```

Parses and stores the header row of the CSV file.

Parameters

line	The header row of the CSV file.

Precondition

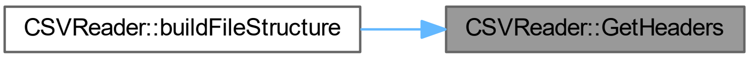
The CSV file is open for reading.

Postcondition

The 'Headers' vector is populated with column headers from the CSV file.

Definition at line 66 of file CSVReader.cpp.

Here is the caller graph for this function:



3.2.3.7 isOpen()

```
bool CSVReader::isOpen ( ) const
```

Checks if the CSV file is open.

Returns

true if the CSV file is open, false otherwise.

Precondition

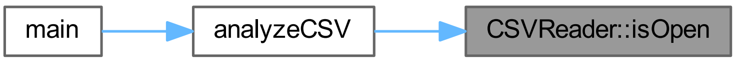
None.

Postcondition

None.

Definition at line 56 of file CSVReader.cpp.

Here is the caller graph for this function:



3.2.3.8 ParseLine()

```
std::vector< std::string > CSVReader::ParseLine (
             const std::string & Record ) [static]
```

Parses a single data row of the CSV file into a Row object.

Parameters

Line	The data row to parse.
r	Reference to the Row object to store the parsed data.

Precondition

The CSV file is open for reading.

Postcondition

The 'r' object is updated with data from the input 'Line'.

Parameters

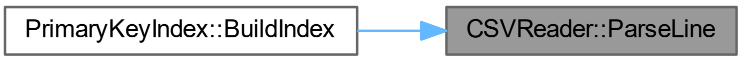
param

Precondition

Postcondition

Definition at line 108 of file CSVReader.cpp.

Here is the caller graph for this function:

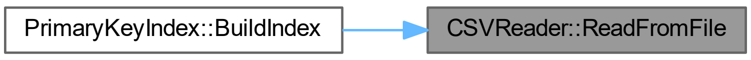


3.2.3.9 ReadFromFile()

```
std::pair< std::size_t, std::string > CSVReader::ReadFromFile (
             std::ifstream & file )  [static]
```

Definition at line 135 of file CSVReader.cpp.

Here is the caller graph for this function:

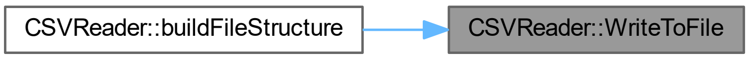


3.2.3.10 WriteToFile()

```
void CSVReader::WriteToFile (
             const std::string & str,
             std::ofstream & file )
```

Definition at line 128 of file CSVReader.cpp.

Here is the caller graph for this function:



3.2.4 Member Data Documentation

3.2.4.1 Headers

```
std::vector<std::string> CSVReader::Headers  [private]
```

Stores the column headers from the CSV file.

Definition at line 117 of file CSVReader.h.

3.2.4.2 ZipCSV

```
std::ifstream CSVReader::ZipCSV  [private]
```

Represents the input CSV file stream used to open and read the CSV file.

Definition at line 116 of file CSVReader.h.

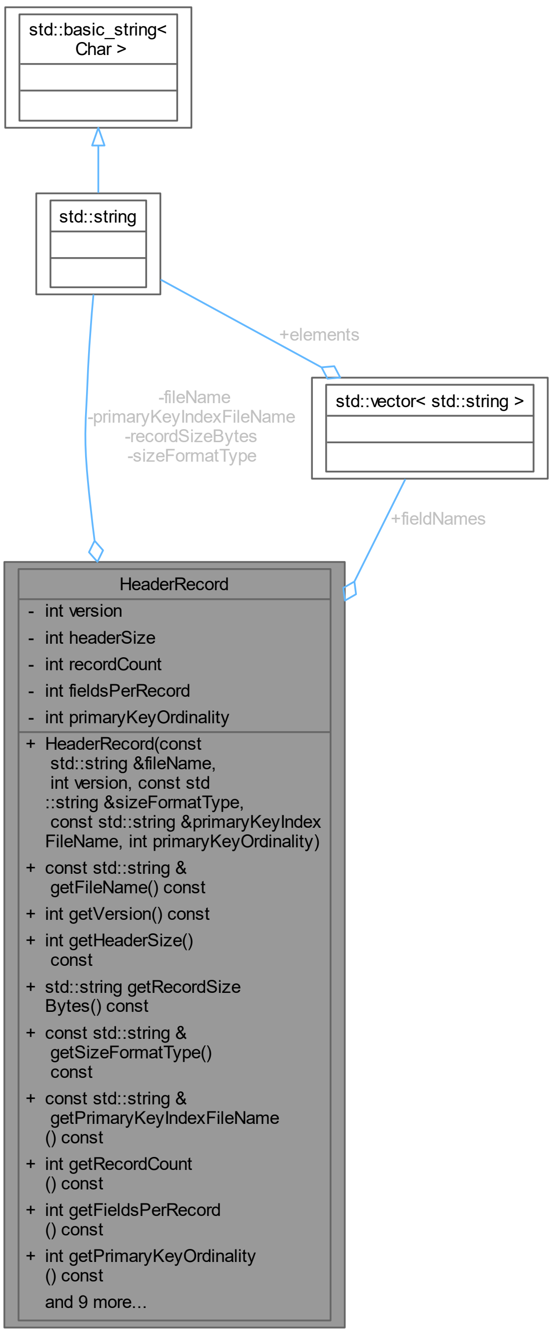
The documentation for this class was generated from the following files:

- CSVReader.h
- CSVReader.cpp

3.3 HeaderRecord Class Reference

```
#include <HeaderRecord.h>
```

Collaboration diagram for HeaderRecord:


std::basic_string< Char >
std::string
+elements
-fileName
-primaryKeyIndexFileName
-recordSizeBytes
-sizeFormatType
std::vector< std::string >
+fieldNames
HeaderRecord
- int version
- int headerSize
- int recordCount
- int fieldsPerRecord
- int primaryKeyOrdinality
+ HeaderRecord(const std::string &fileName, int version, const std ::string &sizeFormatType, const std::string &primaryKeyIndex FileName, int primaryKeyOrdinality)
+ const std::string & getFileName() const
+ int getVersion() const
+ int getHeaderSize() const
+ std::string getRecordSize Bytes() const
+ const std::string & getSizeFormatType() const
+ const std::string & getPrimaryKeyIndexFileName () const
+ int getRecordCount () const
+ int getFieldsPerRecord () const
+ int getPrimaryKeyOrdinality () const
and 9 more...

Public Member Functions

- HeaderRecord (const std::string &fileName, int version, const std::string &sizeFormatType, const std::string &primaryKeyIndexFileName, int primaryKeyOrdinality)
- const std::string & getFileName () const
- int getVersion () const
- int getHeaderSize () const

- std::string getRecordSizeBytes () const
- const std::string & getSizeFormatType () const
- const std::string & getPrimaryKeyIndexFileName () const
- int getRecordCount () const
- int getFieldsPerRecord () const
- int getPrimaryKeyOrdinality () const
- void setFileName (const std::string &newFileName)
- void setVersion (int newVersion)
- void setHeaderSize ()
- void setRecordSizeBytes (std::string newRecordSizeBytes)
- void setSizeFormatType (const std::string &newSizeFormatType)
- void setPrimaryKeyIndexFileName (const std::string &newPrimaryKeyIndexFileName)
- void setRecordCount (int newRecordCount)
- void setFieldsPerRecord (int newFieldsPerRecord)
- void setPrimaryKeyOrdinality (int newPrimaryKeyOrdinality)

Public Attributes

- std::vector< std::string > fieldNames

Private Attributes

- std::string fileName
- int version
- int headerSize
- std::string recordSizeBytes = "variable"
- std::string sizeFormatType
- std::string primaryKeyIndexFileName
- int recordCount
- int fieldsPerRecord
- int primaryKeyOrdinality

3.3.1 Detailed Description

Definition at line 7 of file HeaderRecord.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 HeaderRecord()

```
HeaderRecord::HeaderRecord (
            const std::string & fileName,
            int version,
            const std::string & sizeFormatType,
            const std::string & primaryKeyIndexFileName,
            int primaryKeyOrdinality )
```

Definition at line 3 of file HeaderRecord.cpp.

3.3.3 Member Function Documentation

3.3.3.1 getFieldsPerRecord()

```
int HeaderRecord::getFieldsPerRecord ( ) const
```

Definition at line 46 of file HeaderRecord.cpp.

3.3.3.2 getFileName()

```
const std::string & HeaderRecord::getFileName ( ) const
```

Definition at line 18 of file HeaderRecord.cpp.

3.3.3.3 getHeaderSize()

```
int HeaderRecord::getHeaderSize ( ) const
```

Definition at line 26 of file HeaderRecord.cpp.

3.3.3.4 getPrimaryKeyIndexFileName()

```
const std::string & HeaderRecord::getPrimaryKeyIndexFileName ( ) const
```

Definition at line 38 of file HeaderRecord.cpp.

3.3.3.5 getPrimaryKeyOrdinality()

```
int HeaderRecord::getPrimaryKeyOrdinality ( ) const
```

Definition at line 50 of file HeaderRecord.cpp.

3.3.3.6 getRecordCount()

```
int HeaderRecord::getRecordCount ( ) const
```

Definition at line 42 of file HeaderRecord.cpp.

3.3.3.7 getRecordSizeBytes()

```
std::string HeaderRecord::getRecordSizeBytes ( ) const
```

Definition at line 30 of file HeaderRecord.cpp.

3.3.3.8 getSizeFormatType()

```
const std::string & HeaderRecord::getSizeFormatType ( ) const
```

Definition at line 34 of file HeaderRecord.cpp.

3.3.3.9 getVersion()

```
int HeaderRecord::getVersion ( ) const
```

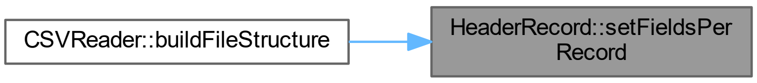
Definition at line 22 of file HeaderRecord.cpp.

3.3.3.10 setFieldsPerRecord()

```
void HeaderRecord::setFieldsPerRecord (
             int newFieldsPerRecord )
```

Definition at line 83 of file HeaderRecord.cpp.

Here is the caller graph for this function:


CSVReader::buildFileStructure
HeaderRecord::setFieldsPer Record

3.3.3.11 setFileName()

```
void HeaderRecord::setFileName (
             const std::string & newFileName )
```

Definition at line 55 of file HeaderRecord.cpp.

3.3.3.12 setHeaderSize()

```
void HeaderRecord::setHeaderSize ( )
```

Definition at line 63 of file HeaderRecord.cpp.

3.3.3.13 setPrimaryKeyIndexFileName()

```
void HeaderRecord::setPrimaryKeyIndexFileName (
             const std::string & newPrimaryKeyIndexFileName )
```

Definition at line 75 of file HeaderRecord.cpp.

3.3.3.14 setPrimaryKeyOrdinality()

```
void HeaderRecord::setPrimaryKeyOrdinality (
            int newPrimaryKeyOrdinality )
```

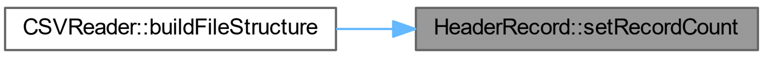
Definition at line 87 of file HeaderRecord.cpp.

3.3.3.15 setRecordCount()

```
void HeaderRecord::setRecordCount (
            int newRecordCount )
```

Definition at line 79 of file HeaderRecord.cpp.

Here is the caller graph for this function:



3.3.3.16 setRecordSizeBytes()

```
void HeaderRecord::setRecordSizeBytes (
            std::string newRecordSizeBytes )
```

Definition at line 67 of file HeaderRecord.cpp.

3.3.3.17 setSizeFormatType()

```
void HeaderRecord::setSizeFormatType (
            const std::string & newSizeFormatType )
```

Definition at line 71 of file HeaderRecord.cpp.

3.3.3.18 setVersion()

```
void HeaderRecord::setVersion (
            int newVersion )
```

Definition at line 59 of file HeaderRecord.cpp.

3.3.4 Member Data Documentation

3.3.4.1 fieldNames

```
std::vector<std::string> HeaderRecord::fieldNames
```

Definition at line 9 of file HeaderRecord.h.

3.3.4.2 fieldsPerRecord

```
int HeaderRecord::fieldsPerRecord  [private]
```

Definition at line 49 of file HeaderRecord.h.

3.3.4.3 fileName

```
std::string HeaderRecord::fileName  [private]
```

Definition at line 42 of file HeaderRecord.h.

3.3.4.4 headerSize

```
int HeaderRecord::headerSize  [private]
```

Definition at line 44 of file HeaderRecord.h.

3.3.4.5 primaryKeyIndexFileName

```
std::string HeaderRecord::primaryKeyIndexFileName  [private]
```

Definition at line 47 of file HeaderRecord.h.

3.3.4.6 primaryKeyOrdinality

```
int HeaderRecord::primaryKeyOrdinality  [private]
```

Definition at line 50 of file HeaderRecord.h.

3.3.4.7 recordCount

```
int HeaderRecord::recordCount  [private]
```

Definition at line 48 of file HeaderRecord.h.

3.3.4.8 recordSizeBytes

```
std::string HeaderRecord::recordSizeBytes = "variable"  [private]
```

Definition at line 45 of file HeaderRecord.h.

3.3.4.9 sizeFormatType

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

Definition at line 46 of file HeaderRecord.h.

3.3.4.10 version

```
int HeaderRecord::version  [private]
```

Definition at line 43 of file HeaderRecord.h.

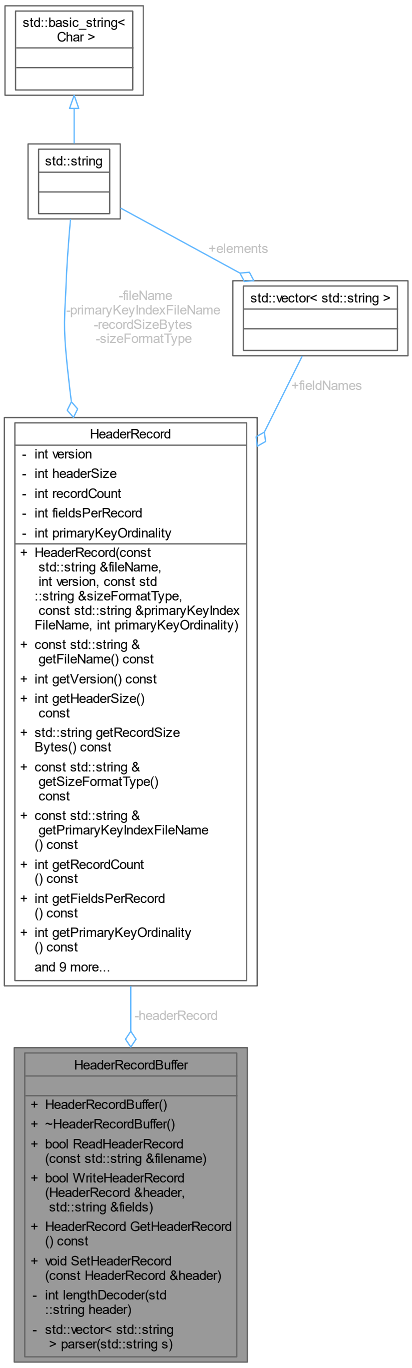
The documentation for this class was generated from the following files:

- HeaderRecord.h
- HeaderRecord.cpp

3.4 HeaderRecordBuffer Class Reference

```
#include <HeaderRecordBuffer.h>
```

Collaboration diagram for HeaderRecordBuffer:


std::basic_string< Char >
std::string
+elements
std::vector< std::string >
-fileName
-primaryKeyIndexFileName
-recordSizeBytes
-sizeFormatType
+fieldNames
HeaderRecord
- int version
- int headerSize
- int recordCount
- int fieldsPerRecord
- int primaryKeyOrdinality
+ HeaderRecord(const std::string &fileName, int version, const std::string &sizeFormatType, const std::string &primaryKeyIndexFileName, int primaryKeyOrdinality)
+ const std::string & getFileName() const
+ int getVersion() const
+ int getHeaderSize() const
+ std::string getRecordSizeBytes() const
+ const std::string & getSizeFormatType() const
+ const std::string & getPrimaryKeyIndexFileName() const
+ int getRecordCount() const
+ int getFieldsPerRecord() const
+ int getPrimaryKeyOrdinality() const
and 9 more...
-headerRecord
HeaderRecordBuffer
+ HeaderRecordBuffer()
+ ~HeaderRecordBuffer()
+ bool ReadHeaderRecord(const std::string &filename)
+ bool WriteHeaderRecord(HeaderRecord &header, std::string &fields)
+ HeaderRecord GetHeaderRecord() const
+ void SetHeaderRecord(const HeaderRecord &header)
- int lengthDecoder(std::string header)
- std::vector< std::string > parser(std::string s)

Public Member Functions

- HeaderRecordBuffer ()

  Constructor for the HeaderRecordBuffer class.
- ∼HeaderRecordBuffer ()

  destructor
- bool ReadHeaderRecord (const std::string &filename)

Method to read file and store header record data.
- bool WriteHeaderRecord (HeaderRecord &header, std::string &fields)

    Method to write a header record to a file.
- HeaderRecord GetHeaderRecord () const

    Method to get the HeaderRecord object within the HeaderRecordBuffer.
- void SetHeaderRecord (const HeaderRecord &header)

    method to grab data and set to the HeaderRecordBuffer

Private Member Functions

- int lengthDecoder (std::string header)

    Method to format the sie format type.
- std::vector< std::string > parser (std::string s)

    Method for parsing the header record.

Private Attributes

- HeaderRecord headerRecord

3.4.1 Detailed Description

Definition at line 21 of file HeaderRecordBuffer.h.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 HeaderRecordBuffer()

```
HeaderRecordBuffer::HeaderRecordBuffer ( )
```

Constructor for the HeaderRecordBuffer class.

Constructor definition.

Parameters

none	N/A

Precondition

none

Postcondition

The ReadHeaderRecord object is constructed

Definition at line 30 of file HeaderRecordBuffer.cpp.

3.4.2.2 ∼HeaderRecordBuffer()

```
HeaderRecordBuffer::∼HeaderRecordBuffer ( )
```

destructor

Method to destroy buffer.

Parameters

none	N/A

Precondition

None.

Postcondition

The ReadHeaderRecord object is destroyed

Parameters

None	N/A

Precondition

None

Postcondition

The ReadHeaderRecordBuffer object is destroyed

Definition at line 225 of file HeaderRecordBuffer.cpp.

3.4.3 Member Function Documentation

3.4.3.1 GetHeaderRecord()

```
HeaderRecord HeaderRecordBuffer::GetHeaderRecord ( ) const
```

Method to get the HeaderRecord object within the HeaderRecordBuffer.

Method to get headerRecord object headerRecord.

Parameters

none	N/A

Precondition

none

Postcondition

HeaderRecord should be returned

Parameters

none	N/A

Precondition

None.

Postcondition

headerRecord struct is returned

Definition at line 205 of file HeaderRecordBuffer.cpp.

3.4.3.2 lengthDecoder()

```
int HeaderRecordBuffer::lengthDecoder (
            std::string header )  [private]
```

Method to format the sie format type.

Method to format length indicators.

Parameters

header	a comma separated string for field names

Precondition

None.

Postcondition

an integer will be returned

Parameters

header	comma separated string

Precondition

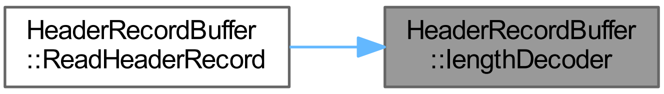
None

Postcondition

integer is returned for record size

Definition at line 235 of file HeaderRecordBuffer.cpp.

Here is the caller graph for this function:


HeaderRecordBuffer ::ReadHeaderRecord
HeaderRecordBuffer ::lengthDecoder

3.4.3.3 parser()

```
std::vector< std::string > HeaderRecordBuffer::parser (
            std::string s )  [private]
```

Method for parsing the header record.

Method to parse record.

Parameters

s	A comma separated string

Precondition

None.

Postcondition

a vector is returned with all comma separated word in their own spot

Parameters

s	comma separated string object

Precondition

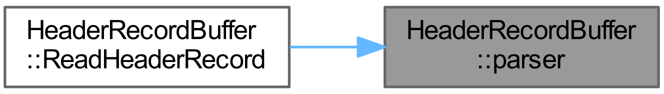
The file must exist

Postcondition

a vector of the comma separated values inside vector

Definition at line 274 of file HeaderRecordBuffer.cpp.

Here is the caller graph for this function:


HeaderRecordBuffer ::ReadHeaderRecord
HeaderRecordBuffer ::parser

3.4.3.4 ReadHeaderRecord()

```
bool HeaderRecordBuffer::ReadHeaderRecord (
            const std::string & filename )
```

Method to read file and store header record data.

Parameters

filename	The name of the CSV file to open.

Precondition

The file must exist

Postcondition

The ReadHeaderRecord object is constructed

Parameters

filename	The name of the CSV file to open.

Precondition

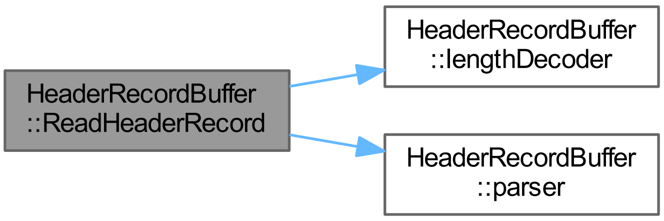
The file must exist

Postcondition

The ReadHeaderRecord object is constructed and stored

Definition at line 41 of file HeaderRecordBuffer.cpp.

Here is the call graph for this function:


HeaderRecordBuffer ::ReadHeaderRecord
HeaderRecordBuffer ::lengthDecoder
HeaderRecordBuffer ::parser

3.4.3.5 SetHeaderRecord()

```
void HeaderRecordBuffer::SetHeaderRecord (
             const HeaderRecord & header )
```

method to grab data and set to the HeaderRecordBuffer

Method to set headerRecord.

Parameters

header	HeaderRecord object made by some user or from file

Precondition

None.

Postcondition

headerRecord inside HeaderRecordBuffer object should contain contain the data of the HeaderRecord object

Parameters

header	HeaderRecord object made with desired data

Precondition

None

Postcondition

headerRecord now has the same values as header

Definition at line 215 of file HeaderRecordBuffer.cpp.

3.4.3.6 WriteHeaderRecord()

```
bool HeaderRecordBuffer::WriteHeaderRecord (
             HeaderRecord & header,
             std::string & fields )
```

Method to write a header record to a file.

Method to write file.

Parameters

header	A headerRecord object
fields	The fields of the file

Precondition

None.

Postcondition

A file is created with length indicated records in bytes

Parameters

header	HeaderRecord object with desired data

Precondition

None

Postcondition

A new file will be written and names according to header's filename

Definition at line 103 of file HeaderRecordBuffer.cpp.

3.4.4 Member Data Documentation

3.4.4.1 headerRecord

```
HeaderRecord HeaderRecordBuffer::headerRecord  [private]
```

Definition at line 77 of file HeaderRecordBuffer.h.

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

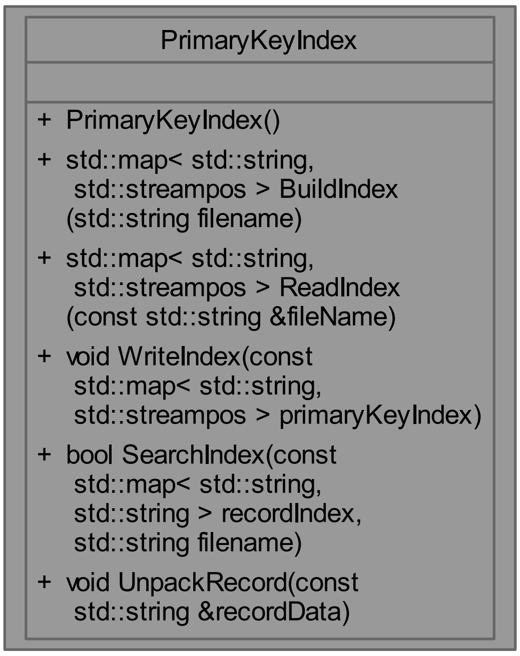
- HeaderRecordBuffer.h
- HeaderRecordBuffer.cpp

3.5 PrimaryKeyIndex Class Reference

Represents the Primary Key Index functionality. This class provides methods for building, reading, writing, searching, and unpacking a primary key index.

```
#include <PrimaryKeyIndex.h>
```

Collaboration diagram for PrimaryKeyIndex:


PrimaryKeyIndex
+ PrimaryKeyIndex()
+ std::map< std::string, std::streampos > BuildIndex (std::string filename)
+ std::map< std::string, std::streampos > ReadIndex (const std::string &fileName)
+ void WriteIndex(const std::map< std::string, std::streampos > primaryKeyIndex)
+ bool SearchIndex(const std::map< std::string, std::string > recordIndex, std::string filename)
+ void UnpackRecord(const std::string &recordData)

Public Member Functions

- PrimaryKeyIndex ()

  Constructor to initialize the PrimaryKeyIndex.
- std::map< std::string, std::streampos > BuildIndex (std::string filename)

  Builds the primary key index.
- std::map< std::string, std::streampos > ReadIndex (const std::string &fileName)

  Reads the primary key index from a file.
- void WriteIndex (const std::map< std::string, std::streampos > primaryKeyIndex)

  Writes the primary key index to a file.
- bool SearchIndex (const std::map< std::string, std::string > recordIndex, std::string filename)

  Searches for a record in the index using a primary key.
- void UnpackRecord (const std::string &recordData)

  Unpacks and displays a record given its data.

3.5.1 Detailed Description

Represents the Primary Key Index functionality. This class provides methods for building, reading, writing, searching, and unpacking a primary key index.

Definition at line 28 of file PrimaryKeyIndex.h.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 PrimaryKeyIndex()

```
PrimaryKeyIndex::PrimaryKeyIndex ( )
```

Constructor to initialize the PrimaryKeyIndex.

Default constructor for PrimaryKeyIndex.

Precondition

None.

Postcondition

The PrimaryKeyIndex object is constructed.

Initializes any member variables if necessary.

Definition at line 17 of file PrimaryKeyIndex.cpp.

3.5.3 Member Function Documentation

3.5.3.1 BuildIndex()

```
std::map< std::string, std::streampos > PrimaryKeyIndex::BuildIndex (
            std::string filename )
```

Builds the primary key index.

Builds a primary key index based on a given file.

Parameters

filename	Name of the file to build the index from.

Returns

A map representing the primary key index.

Precondition

The file with the given filename exists and contains valid data.

Postcondition

The primary key index is built and returned.

Parameters

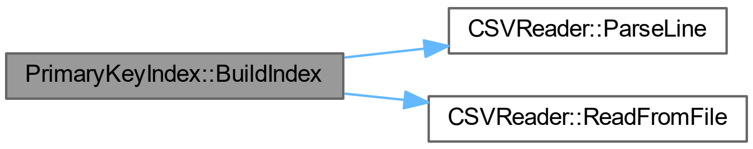
filename	The name of the file to be indexed.

Returns

A map representing the primary key index.

Definition at line 26 of file PrimaryKeyIndex.cpp.

Here is the call graph for this function:



3.5.3.2 ReadIndex()

```
std::map< std::string, std::streampos > PrimaryKeyIndex::ReadIndex (
            const std::string & fileName )
```

Reads the primary key index from a file.

Reads a primary key index from a given file.

Parameters

fileName	Name of the file to read the index from.

Returns

A map representing the primary key index.

Precondition

The file with the given filename exists and contains a valid index.

Postcondition

The primary key index is read and returned.

Parameters

fileName	The name of the index file.

Returns

A map representing the primary key index.

Definition at line 59 of file PrimaryKeyIndex.cpp.

3.5.3.3 SearchIndex()

```
bool PrimaryKeyIndex::SearchIndex (
            const std::map< std::string, std::string > recordIndex,
            std::string filename )
```

Searches for a record in the index using a primary key.

Searches for a record in the primary key index.

Parameters

recordIndex	The map of records to search within.
filename	The name of the file containing the primary key index.

Returns

True if the record is found, otherwise false.

Precondition

The file with the given filename exists and contains a valid index.

Postcondition

None.

Parameters

recordIndex	A map representing the primary key index.
filename	The name of the file where the records are stored.

Returns

True if the record is found, false otherwise.

Definition at line 116 of file PrimaryKeyIndex.cpp.

3.5.3.4 UnpackRecord()

```
void PrimaryKeyIndex::UnpackRecord (
             const std::string & recordData )
```

Unpacks and displays a record given its data.

Unpacks and displays a given record.

Parameters

recordData	The data of the record to unpack and display.

Precondition

None.

Postcondition

The record data is unpacked and displayed.

Parameters

recordData	The record data as a string.

Precondition

The recordData string is correctly formatted.

Postcondition

The record is unpacked and displayed to the console.

Definition at line 143 of file PrimaryKeyIndex.cpp.

3.5.3.5 WriteIndex()

```
void PrimaryKeyIndex::WriteIndex (
             const std::map< std::string, std::streampos > primaryKeyIndex )
```

Writes the primary key index to a file.

Parameters

primaryKeyIndex	The primary key index to write.

Precondition

None.

Postcondition

The primary key index is written to a file.

Parameters

primaryKeyIndex	A map representing the primary key index.

Precondition

The map primaryKeyIndex is correctly populated.

Postcondition

The index is written to the file "KeyIndex.txt".

Definition at line 93 of file PrimaryKeyIndex.cpp.

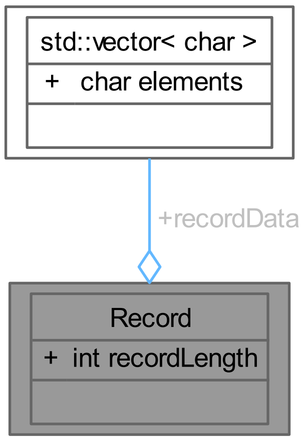
The documentation for this class was generated from the following files:

- PrimaryKeyIndex.h
- PrimaryKeyIndex.cpp

3.6 Record Struct Reference

```
#include <CSVReader.h>
```

Collaboration diagram for Record:



Public Attributes

- int recordLength
- std::vector< char > recordData

3.6.1 Detailed Description

Definition at line 50 of file CSVReader.h.

3.6.2 Member Data Documentation

3.6.2.1 recordData

```
std::vector<char> Record::recordData
```

Definition at line 52 of file CSVReader.h.

3.6.2.2 recordLength

```
int Record::recordLength
```

Definition at line 51 of file CSVReader.h.

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

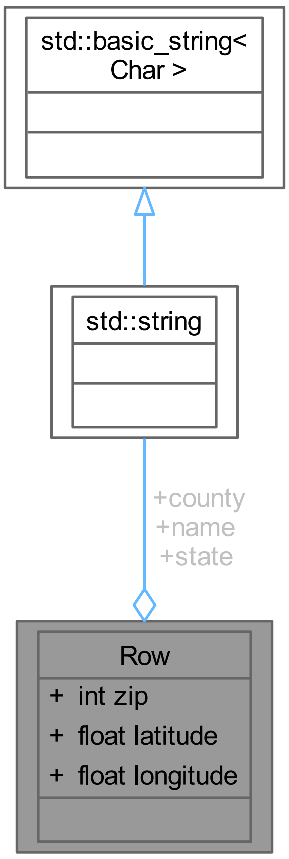
- CSVReader.h

3.7 Row Struct Reference

Represents a row of data in the CSV file. This struct stores information for a single row of data in the CSV file, including the ZIP code, name, state, county, latitude, and longitude.

```
#include <CSVReader.h>
```

Collaboration diagram for Row:



Public Attributes

- int zip
- std::string name
- std::string state
- std::string county
- float latitude
- float longitude

3.7.1 Detailed Description

Represents a row of data in the CSV file. This struct stores information for a single row of data in the CSV file, including the ZIP code, name, state, county, latitude, and longitude.

Definition at line 41 of file CSVReader.h.

3.7.2 Member Data Documentation

3.7.2.1 county

```
std::string Row::county
```

The county.

Definition at line 45 of file CSVReader.h.

3.7.2.2 latitude

```
float Row::latitude
```

The latitude.

Definition at line 46 of file CSVReader.h.

3.7.2.3 longitude

```
float Row::longitude
```

The longitude.

Definition at line 47 of file CSVReader.h.

3.7.2.4 name

```
std::string Row::name
```

The place name.

Definition at line 43 of file CSVReader.h.

3.7.2.5 state

```
std::string Row::state
```

The state.

Definition at line 44 of file CSVReader.h.

3.7.2.6 zip

```
int Row::zip
```

The ZIP code.

Definition at line 42 of file CSVReader.h.

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

- CSVReader.h

Chapter 4

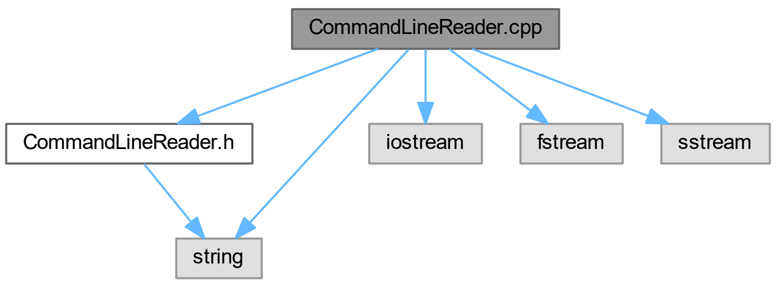
File Documentation

4.1 CommandLineReader.cpp File Reference

Member function definitions for class CommandLineReader.

```
#include "CommandLineReader.h"
#include <string>
#include <iostream>
#include <fstream>
#include <sstream>
```

Include dependency graph for CommandLineReader.cpp:



4.1.1 Detailed Description

Member function definitions for class CommandLineReader.

Author

Abdirahman Abdi

See also

CommandLineReader.h for declaration.

This class provides functionality to:

- Display a menu to the user.
- Accept commands from the user.
- Search for a given ZIP code in a database.
- Find the ZIP code of a place based on its name and latitude.
- Exit the application.

Assumptions:

- The database file 'us_postal_codes.txt' is properly formatted and available.
- The user provides valid input.

Definition in file CommandLineReader.cpp.

4.2 CommandLineReader.cpp

Go to the documentation of this file.

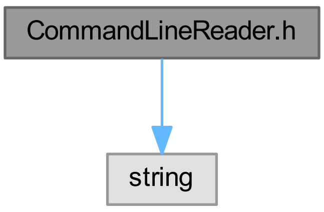
```

#include "CommandLineReader.h"
#include <string>
#include <iostream>
#include <fstream>
#include <sstream>

CommandLineReader::CommandLineReader() {
    running = true;
}

void CommandLineReader::Main() {
    std::cout « "\n-- welcome to command reader --\n\n";
    while (running) {
        std::cout « "press E to enter zip code and see if it’s in database\n";
        std::cout « "press F to find and print zipcode\n";
        std::cout « "press T to close program\n";

        std::string input;
        std::cin » input;

        ParseCommandLine(input);
    }
}

void CommandLineReader::ParseCommandLine(const std::string& input) {
    std::string line, fileZip, fileCity, state, county, latitude, longitude;
    bool found = false;

    if (input == "E" || input == "e") {
        std::cout « "Enter the zip code: ";
        std::string enteredZip;  // Use a different variable to capture user input
        std::cin » enteredZip;
        std::ifstream file("us_postal_codes.txt"); // reead file
        while (getline(file, line)) { // goes through all the lines in us_postal_codes.csv
            std::istringstream iss(line); //reading across the line using ’,’ as breakpoint
            getline(iss, fileZip, ’,’);
            if (fileZip == enteredZip) {
                std::cout « "Zip code " « enteredZip « " is in the database." « std::endl;
                found = true;
                break; //break the reading loop
            }
        }
        if (!found) { // if loop is done but still no zip file then come here and print
            std::cout « "Zip code " « enteredZip « " is not in the database." « std::endl;
        }
        //close file
        file.close();

    }
    else if (input == "F" || input == "f") {
        //finding zip of place from the name
        std::cout « "Enter the place name to find its zip code: e.g 'Amherst' ";
        std::string enteredCity;  // read user entered city
        std::cin » enteredCity;
        std::cout « "Enter the place latitude to find its zip code: e.g '42.3671' ";
        std::string enteredLat;  // read user entered latitude
        std::cin » enteredLat;

        std::ifstream file("us_postal_codes.txt"); //read from database
        while (getline(file, line)) {
            std::istringstream iss(line); // go through each line
            // go across the line with breakpoints ',' and input each data into string format and
      their relevant variables
            getline(iss, fileZip, ',');
            getline(iss, fileCity, ',');  // save cityname in strings with break point ','
            getline(iss, state, ',');
            getline(iss, county, ',');
            getline(iss, latitude, ','); // save latitude
            getline(iss, longitude, ',');
            if (fileCity == enteredCity && latitude == enteredLat ) { // if for accuracy both cityname
      and latitude point match then
                std::cout « "Zip code for " « enteredCity « " @Latitude: " « enteredLat « " is: " «
      fileZip « std::endl;
                found = true;
                break;
            }
        }
        if (!found) {
            std::cout « "City of " « enteredCity « " or its latitude @: " « enteredLat «" not found in
      the database." « std::endl;
        }
        file.close(); //close file read buffer

    } else if (input == "T" || input == "t") {
        running = false;  // Exit the loop
    } else {
        std::cout « "Invalid input. Please try again.\n";
    }
}



```

4.3 CommandLineReader.h File Reference

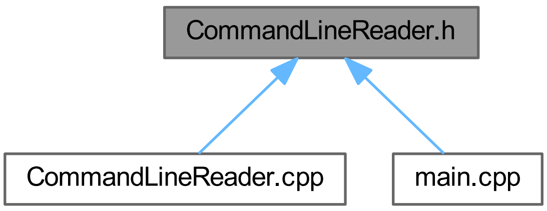
Declarations for class CommandLineReader.

`#include <string>`

Include dependency graph for CommandLineReader.h:



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



Classes

- class CommandLineReader

4.3.1 Detailed Description

Declarations for class CommandLineReader.

Author

Abdirahman Abdi

See also

CommandLineReader.cpp for the implementation of these functions.

This file declares the class CommandLineReader, which provides functionality to read and process commands from the command line. The class includes member functions for starting the reader loop and parsing command line input.

Assumptions:

- Input from the command line is provided in a valid format.
- Parsing functions are capable of handling various types of command input.
- The loop continues until a termination condition is met.

Definition in file CommandLineReader.h.

4.4 CommandLineReader.h

Go to the documentation of this file.

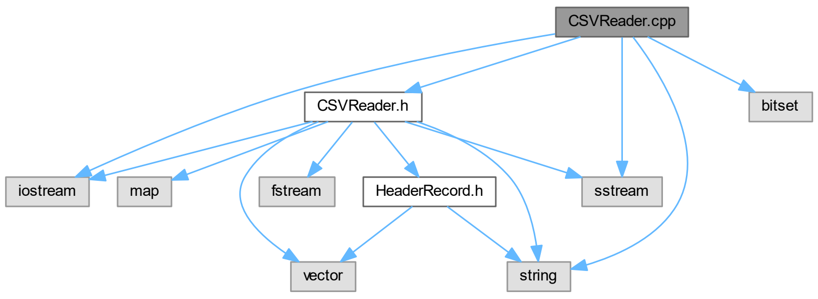
```

#ifndef COMMANDLINEREADER_H
#define COMMANDLINEREADER_H

#include <string>

class CommandLineReader {
public:
    CommandLineReader();

    void Main();

    void ParseCommandLine(const std::string& input);

private:
    bool running;
};

#endif //COMMANDLINEREADER_H
```

4.5 CSVReader.cpp File Reference

Member function definitions for class CSVReader.

```
#include "CSVReader.h"
#include <iostream>
#include <string>
#include <sstream>
#include <bitset>
```

Include dependency graph for CSVReader.cpp:



4.5.1 Detailed Description

Member function definitions for class CSVReader.

Author

Fabian MullerDahlberg

Authors

(comments by Hamaad) (Testing done by Shishir) (Doxygen documentation by Abdi)

See also

CCSVReader.h for declaration.

- Constructor: Opens a specified CSV file for reading.
- isOpen(): Checks if the CSV file is currently open.
- GetHeaders(): Parses and stores the header row of the CSV file, populating the Headers vector with column headers.
- ReadFile(): Reads and processes the entire CSV file, including parsing data rows and calculating state statistics.
- ParseLine(): Parses a single data row of the CSV file into a Row object, updating it with data from the input line.
- CheckMaxima(): Checks and updates a map (StateMaximums) with maximum and minimum values for latitude and longitude based on the input Row.
- CompareExtremes(): Compares and updates the maximum and minimum values for latitude and longitude in a state.
- GetStateMaximums(): Retrieves a copy of the StateMaximums map, which contains state statistics.
- close(): Closes the CSV file if it's currently open.

Assumptions:

- The input CSV file is properly formatted with valid data.
- The CSV file has a header row that defines column names.
- Latitude and longitude values are provided in decimal format.
- The CSV file contains data for multiple states.
- The CSV file follows the format: Zip,Name,State,County,Latitude,Longitude.
- Rows with missing or invalid data will be skipped.
- The CSV file may be large, so memory usage is considered.
- State statistics, including maximum and minimum values, are calculated and stored for each state in the data.

Definition in file CSVReader.cpp.

4.6 CSVReader.cpp

Go to the documentation of this file.

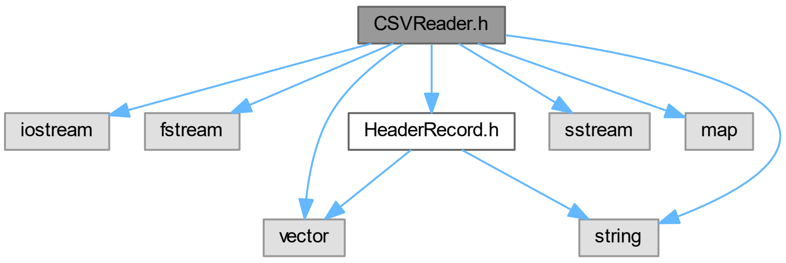
```cpp

#include "CSVReader.h"
#include <iostream>
#include <string>
#include <sstream>
#include <bitset>

CSVReader::CSVReader(const std::string filename) {
    ZipCSV.open(filename, std::ios::in);
}

bool CSVReader::isOpen() const {
    return ZipCSV.is_open();
}

void CSVReader::GetHeaders(std::string &line) {
    std::stringstream stream(line);
    std::string header;
    // Parses the first of the csv by comma to get headers
    while (std::getline(stream, header, ',')) {
        // Adds headers to the vector
        Headers.push_back(header);
    }
}

void CSVReader::buildFileStructure(std::ofstream& file,HeaderRecord& headerRecord) {
    std::string line;

    // Read and store the header row of the CSV file.
    std::getline(ZipCSV, line, '\n');
    GetHeaders(line);
    headerRecord.setFieldsPerRecord(Headers.size());
    int recordCount = 0;
    // Read and process each data row of the CSV file.
    while (std::getline(ZipCSV, line, '\n')) {

        // line needs to be converted to length indicated
        // after converting read to file.
       //  ParseLine(line, NewRow);
        WriteToFile(line, file);
        recordCount = recordCount + 1;
    }
    headerRecord.setRecordCount(recordCount);
}

std::vector<std::string> CSVReader::ParseLine(const std::string& Record) {
    // for parsing a length indicated record after being read from data file, not csv
    // consider switching to the c++ » operator
    // parse by comma based on fields read from header
    std::stringstream stream(Record);
    std::string field;
    std::vector<std::string> parsedRecord;
    while (std::getline(stream, field, ',')) {
        parsedRecord.push_back(field);
    }
    return parsedRecord;
}

// -----------------New methods for handling length-indicated records -------------------------------//
void CSVReader::ConvertToLength(const std::string& inputRecord, Record& outputRecord) {
    // Implementation for ConvertToLength
    // Maintains comma seperated nature of original data but prepends binary integer representing length
    // store row in records struct made up of a char vector and an integer size.
}

void CSVReader::WriteToFile(const std::string& str, std::ofstream& file) {
    // should write a record with its length concatenated to the beginning.
    std::size_t size = str.size();
    file.write(reinterpret_cast<const char*>(&size), sizeof(size));
    file.write(str.c_str(), size);
}

std::pair<std::size_t, std::string> CSVReader::ReadFromFile(std::ifstream& file) {
    // reads the record length to determine offset.
    // watch out fo errors caused by over, or under reading
        // ie. make sure the length of the "length indicator" itself is accounted for
    std::size_t size;
    file.read(reinterpret_cast<char*>(&size), sizeof(size));
    std::string str;
    str.resize(size);
    file.read(&str[0], size);
    //return str;
    return std::make_pair(size, str);
}

//HeaderRecord CSVReader::GenerateHeaderRecord() {
//    HeaderRecord header;
//    // Initialize header fields
//    // may be
//    return header;
//

bool CSVReader::BuildDataFile(const std::string& sourceFile1, const std::string& sourceFile2, const
std::string& destinationFile) {
    // Implementation for BuildDataFile
    // after header record is built use data to define and create file.
    std::ifstream file1("header.txt",std::ios::binary);
    std::ifstream file2("output.txt",std::ios::binary);
    std::ofstream destFile("combined_file.txt",std::ios::binary);

    if (!file1.is_open() || !file2.is_open() || !destFile.is_open()) {
        std::cerr « "Failed to open one or more files." « std::endl;
        return false;
    }

    destFile « file1.rdbuf() « file2.rdbuf();

    file1.close();
    file2.close();
    destFile.close();

    return true;
}

void CSVReader::close() {
    if (ZipCSV.is_open()) {
        ZipCSV.close();
    }
}

CSVReader::~CSVReader(){}
```

4.7 CSVReader.h File Reference

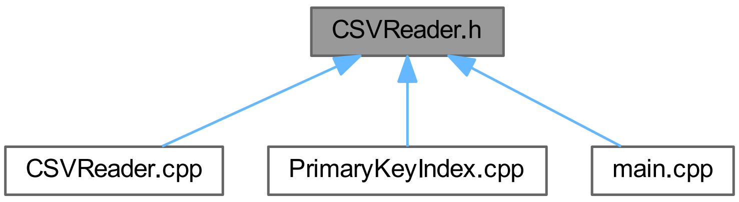
Declarations for class CSVReader.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>
#include <map>
#include "HeaderRecord.h"
```

Include dependency graph for CSVReader.h:



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



Classes

- struct Row

    Represents a row of data in the CSV file. This struct stores information for a single row of data in the CSV file, including the ZIP code, name, state, county, latitude, and longitude.
- struct Record
- class CSVReader

4.7.1 Detailed Description

Declarations for class CSVReader.

Author

Fabian MullerDahlberg

(Comments by Roshan and Fabian) (Testing done by Shishir) (Doxygen documentation by Abdi)

See also

CSVReader.cpp for the implementation of these functions.

This file declares the class CSVReader, which provides functionality to read and process CSV files. The class includes member functions for opening, reading, and analyzing CSV files, as well as storing and retrieving state statistics.

Assumptions:

- The input CSV file is properly formatted with valid data.
- The CSV file has a header row that defines column names.
- Latitude and longitude values are provided in decimal format.
- The CSV file contains data for multiple states.
- The CSV file follows the format: Zip,Name,State,County,Latitude,Longitude.
- Rows with missing or invalid data will be skipped.
- The CSV file may be large, so memory usage is considered.
- State statistics, including maximum and minimum values, are calculated and stored for each state in the data.

Definition in file CSVReader.h.

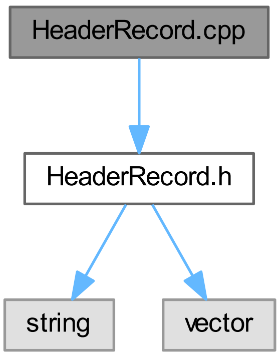
4.8 CSVReader.h

Go to the documentation of this file.

```
#ifndef ZIPCODES_CSVREADER_H
#define ZIPCODES_CSVREADER_H


#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>
#include <map>
#include "HeaderRecord.h"

struct Row {
    int zip;
    std::string name;
    std::string state;
    std::string county;
    float latitude;
    float longitude;
};

struct Record {
    int recordLength;
    std::vector<char> recordData;
    // Add other fields specific to a length-indicated record
};

class CSVReader {
public:

    CSVReader(const std::string filename);

    bool isOpen() const;

    void GetHeaders(std::string &line);

    void buildFileStructure(std::ofstream& file,HeaderRecord& headerRecord);

    static std::vector<std::string> ParseLine(const std::string &line);

    void close();

    ~CSVReader();

    //---------------- New methods for handling length-indicated records ------------------------------------//
    void ConvertToLength(const std::string& inputRecord, Record& outputRecord);
    void WriteToFile(const std::string& str, std::ofstream& file);
    static std::pair<std::size_t, std::string> ReadFromFile(std::ifstream& file);
    HeaderRecord GenerateHeaderRecord();
    bool BuildDataFile(const std::string& sourceFile1, const std::string& sourceFile2, const std::string& destinationFile);

private:
    std::ifstream ZipCSV;
    std::vector<std::string> Headers;
};

#endif //ZIPCODES_CSVREADER_H
```

4.9 HeaderRecord.cpp File Reference

`#include "HeaderRecord.h"`
Include dependency graph for HeaderRecord.cpp:



4.10 HeaderRecord.cpp

Go to the documentation of this file.

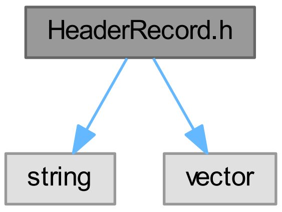
```
#include "HeaderRecord.h"

HeaderRecord::HeaderRecord(
        const std::string& fileName,
        int version,
        const std::string& sizeFormatType,
        const std::string& primaryKeyIndexFileName,
        int primaryKeyOrdinality
)
        : fileName(fileName),
          version(version),
          sizeFormatType(sizeFormatType),
          primaryKeyIndexFileName(primaryKeyIndexFileName),
          primaryKeyOrdinality(primaryKeyOrdinality) {
}

// Implement getter methods
const std::string& HeaderRecord::getFileName() const {
    return fileName;
}

int HeaderRecord::getVersion() const {
    return version;
}

int HeaderRecord::getHeaderSize() const {
    return headerSize;
}

std::string HeaderRecord::getRecordSizeBytes() const {
    return recordSizeBytes;
}

const std::string& HeaderRecord::getSizeFormatType() const {
    return sizeFormatType;
}

const std::string& HeaderRecord::getPrimaryKeyIndexFileName() const {
    return primaryKeyIndexFileName;
}

```

```
int HeaderRecord::getRecordCount() const {
    return recordCount;
}

int HeaderRecord::getFieldsPerRecord() const {
    return fieldsPerRecord;
}

int HeaderRecord::getPrimaryKeyOrdinality() const {
    return primaryKeyOrdinality;
}

// Implement setter methods
void HeaderRecord::setFileName(const std::string& newFileName) {
    fileName = newFileName;
}

void HeaderRecord::setVersion(int newVersion) {
    version = newVersion;
}

void HeaderRecord::setHeaderSize() {
    headerSize = sizeof(*this);
}

void HeaderRecord::setRecordSizeBytes(std::string newRecordSizeBytes) {
    recordSizeBytes = newRecordSizeBytes;
}

void HeaderRecord::setSizeFormatType(const std::string& newSizeFormatType) {
    sizeFormatType = newSizeFormatType;
}

void HeaderRecord::setPrimaryKeyIndexFileName(const std::string& newPrimaryKeyIndexFileName) {
    primaryKeyIndexFileName = newPrimaryKeyIndexFileName;
}

void HeaderRecord::setRecordCount(int newRecordCount) {
    recordCount = newRecordCount;
}

void HeaderRecord::setFieldsPerRecord(int newFieldsPerRecord) {
    fieldsPerRecord = newFieldsPerRecord;
}

void HeaderRecord::setPrimaryKeyOrdinality(int newPrimaryKeyOrdinality) {
    primaryKeyOrdinality = newPrimaryKeyOrdinality;
}
```

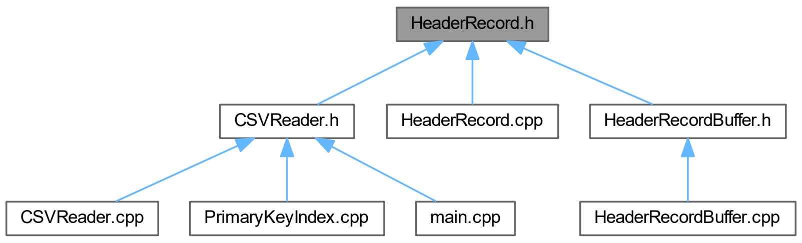
4.11 HeaderRecord.h File Reference

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

Include dependency graph for HeaderRecord.h:



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



Classes

- class HeaderRecord

4.12 HeaderRecord.h

Go to the documentation of this file.

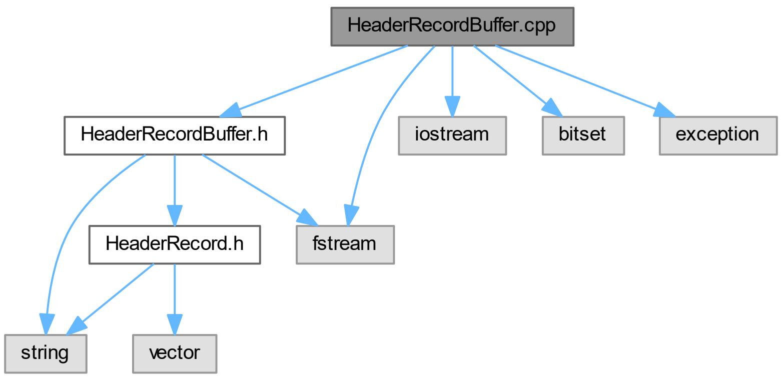
```
#ifndef ZIPCODES_HEADERRECORD_H
#define ZIPCODES_HEADERRECORD_H

#include <string>
#include <vector>

class HeaderRecord {
public:
    std::vector<std::string> fieldNames;
    HeaderRecord(
            const std::string& fileName,
            int version,
            const std::string& sizeFormatType,
            const std::string& primaryKeyIndexFileName,
            int primaryKeyOrdinality
    );

    // Getter methods
    const std::string& getFileName() const;
    int getVersion() const;
    int getHeaderSize() const;
    std::string getRecordSizeBytes() const;
    const std::string& getSizeFormatType() const;
    const std::string& getPrimaryKeyIndexFileName() const;
    int getRecordCount() const;
    int getFieldsPerRecord() const;
    int getPrimaryKeyOrdinality() const;

    // Setter methods
    void setFileName(const std::string& newFileName);
    void setVersion(int newVersion);
    void setHeaderSize();
    void setRecordSizeBytes(std::string newRecordSizeBytes);
    void setSizeFormatType(const std::string& newSizeFormatType);
    void setPrimaryKeyIndexFileName(const std::string& newPrimaryKeyIndexFileName);
    void setRecordCount(int newRecordCount);
    void setFieldsPerRecord(int newFieldsPerRecord);
    void setPrimaryKeyOrdinality(int newPrimaryKeyOrdinality);


private:
    std::string fileName;
    int version;
    int headerSize;
    std::string recordSizeBytes = "variable";
    std::string sizeFormatType;
    std::string primaryKeyIndexFileName;
    int recordCount;
    int fieldsPerRecord;
    int primaryKeyOrdinality;

};

#endif //ZIPCODES_HEADERRECORD_H
```

4.13 HeaderRecordBuffer.cpp File Reference

Definitions for class HeaderRecordBuffer.

```
#include "HeaderRecordBuffer.h"
#include <fstream>
#include <iostream>
#include <bitset>
#include <exception>
```

Include dependency graph for HeaderRecordBuffer.cpp:



4.13.1 Detailed Description

Definitions for class HeaderRecordBuffer.

Author

Fabian MullerDahlberg & Chakong

(Comments by Chakong Lor)(Doxygen documentation by Chakong Lor)

See also

HeaderRecordBuffer.h for the implementation of these functions.

This file defines the class HeaderRecordBuffer, which provides functionality to read and write header records for files. The class includes member functions for writing, reading, and analyzing CSV files.

Definition in file HeaderRecordBuffer.cpp.

4.14 HeaderRecordBuffer.cpp

Go to the documentation of this file.

```
// Created by mulle on 10/11/2023.
//
// Edited by Lor on 10/14/2023

#include "HeaderRecordBuffer.h"
#include <fstream>
#include <iostream>
#include <bitset>
#include <exception>

HeaderRecordBuffer::HeaderRecordBuffer() {
    // Constructor, initialize variables if necessary

}

bool HeaderRecordBuffer::ReadHeaderRecord(const std::string& filename) {
    // Implement the code to read the header record from a file
    // Parse the fields and store them in the headerRecord struct
    // Return true if successful, false if there was an error

    // Method requirements:
    // 1. The file being read must be length indicated

        headerRecord.fileName = filename;
        std::ifstream file(filename);
        std::string header;
        std::string word = "";
        int count = 0;
        std::vector<std::string> vec;

        if (!file.is_open())
        {
            std::cout « "No such file";
            return false;
        }
        while(file » word)
        {
            count++;
            if (count == 2)
            {
                vec = parser(word);
            }

            if (count == 5)
            {
                header = word;
            }
        }
        headerRecord.version = std::stoi(vec[1]);
        headerRecord.sizeFormatType = vec[3];
        headerRecord.recordCount = count;
        word = "";
        headerRecord.headerSize = lengthDecoder(header);
        for (int i = 3; i < header.length(); i++)
        {
            if (header[i] != ',' && header[i] != '\n')
            {
                word = word + header[i];
            }
            else
            {
                headerRecord.fieldNames.push_back(word);
                word = "";
            }
        }
        headerRecord.fieldNames.push_back(word);
        headerRecord.fieldsPerRecord = headerRecord.fieldNames.size();

    return true;
}

bool HeaderRecordBuffer::WriteHeaderRecord( HeaderRecord& header, std::string& fields) {//
    // Implement the code to write the header record to a file
    // Write the fields of the header struct to the file
    // Return true if successful, false if there was an error

    // Method only outputs:
    // 1. headerSize
    // 2. sizeFormatType
    // 3. fieldsPerRecord
    // 4. fieldNames

    // Method only requires
    // the following to write:
    // 1. HeaderRecord object
    //    - filename
    //    - sizeFormatType {ASCII or binary}
    // 2. field names with comma separation (string)
    //    - ex: "Zip,Place,State...etc"

    try
    {
        if (header.headerSize == 0)
        {
            char c;
            std::string temp = "";
            int holder = 0;
            for (int i = 0; i < fields.size();i++)
            {
                c = fields[i];
                if (c != ',' && c != '\n')
                {
                    holder += sizeof(c);
                    temp = temp + c;
                }
                else if (c == ',' || c == '\n')
                {
                    header.fieldNames.push_back(temp);
                    temp = "";
                }
            }
            header.fieldNames.push_back(temp);
            header.headerSize = holder;
            header.fieldsPerRecord = header.fieldNames.size();
        }
        std::ofstream file(header.fileName);

        if (header.sizeFormatType == "ASCII")
        {
            file « "39filename,version,headersize,sizeFormatType" « std::endl;
            file « header.fileName.size() + sizeof(header.version) + sizeof(header.headerSize) +
header.sizeFormatType.size()
            « header.fileName « "," « header.version « "," « header.headerSize «"," «
header.sizeFormatType « std::endl;
            file « 26 « "recordCount,fieldsPerRecord" « std::endl;
            file « sizeof(int) + sizeof(int)«header.recordCount « ","«header.fieldsPerRecord «
std::endl;
            file « header.headerSize;
        }
        if (header.sizeFormatType == "binary")
        {
            file « std::bitset<8>(39).to_string() « "filename,version,headersize,sizeFormatType" «
std::endl;
            file « std::bitset<8>(header.fileName.size() + sizeof(header.version) +
sizeof(header.headerSize) + header.sizeFormatType.size())
            « header.fileName « "," « header.version « "," « header.headerSize «"," «
header.sizeFormatType « std::endl;
            file « std::bitset<8>(26) « "recordCount,fieldsPerRecord" « std::endl;
            file « std::bitset<8>(sizeof(int) + sizeof(int))«header.recordCount« "," «
header.fieldsPerRecord « std::endl;
            std::string byte = std::bitset<32>(header.headerSize).to_string();
            std::string temp = "";
            int coolInteger = 0;
            for (int i =0; i < byte.size(); i++)
            {
                temp = temp + byte[i];
                if (i == 7 || i == 15 || i == 23)
                {
                    temp = "";
                }
            }
            byte = temp;
            file « byte;
        }
        for (int i = 0; i < header.fieldNames.size();i++)
        {
            if (i == header.fieldNames.size() - 1)
            {
                file « header.fieldNames[i] « '\n';
            }
            else
            {
                file « header.fieldNames[i] « ",";
            }
        }
        throw std::runtime_error("");
    }
    catch(...)
    {
        return false;
    }
    return true;
}

HeaderRecord HeaderRecordBuffer::GetHeaderRecord() const {
    return headerRecord;
}

void HeaderRecordBuffer::SetHeaderRecord(const HeaderRecord& header) {
    headerRecord = header;
}

HeaderRecordBuffer::~HeaderRecordBuffer() {
    // Destructor, perform cleanup if necessary
}

int HeaderRecordBuffer::lengthDecoder(std::string header)
{
    std::string temp = "";
    int integer = 0;
    bool binflag = true;
    for (int i = 0; i < header.size(); i++)
    {
        if (isdigit(header[i]))
        {
            temp = temp + header[i];

        }
        if ((header[i] != '0' && header[i] != '1') && isdigit(header[i]))
        {
            binflag = false;
        }
    }

    if (binflag == true)
    {
        //std::cout « temp « std::endl;
        headerRecord.sizeFormatType = "binary";
        integer = std::stoi(temp,0,2);
    }
    if (!binflag)
    {

        headerRecord.sizeFormatType = "ASCII";
        integer = std::stoi(temp);
    }
    return integer;
}

std::vector<std::string> HeaderRecordBuffer::parser(std::string s)
{
    std::string temp = "";
    std::vector<std::string> v;

    for (int i = 0;i < s.size(); i++)
    {
        if(s[i] == ',' || s[i] == '\n')
        {
            v.push_back(temp);
            temp = "";
        }
        else if (s[i] != ',')
        {
            temp = temp + s[i];
        }
    }
    v.push_back(temp);
    return v;
}
```

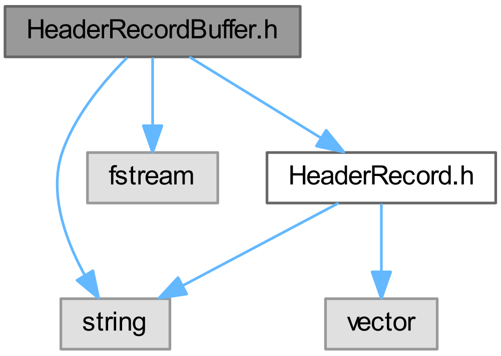
4.15 HeaderRecordBuffer.h File Reference

Declarations for class HeaderRecordBuffer.

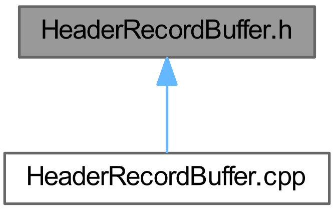
```
#include <string>
#include <fstream>
```

```
#include "HeaderRecord.h"
```

Include dependency graph for HeaderRecordBuffer.h:



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



Classes

- class HeaderRecordBuffer

4.15.1 Detailed Description

Declarations for class HeaderRecordBuffer.

Author

Fabian MullerDahlberg & Chakong

(Comments by Chakong Lor)(Doxygen documentation by Chakong Lor)

See also

HeaderRecordBuffer.cpp for the implementation of these functions.

This file declares the class HeaderRecordBuffer, which provides functionality to read and write header records for files. The class includes member functions for writing, reading, and analyzing CSV files.

Definition in file HeaderRecordBuffer.h.

4.16 HeaderRecordBuffer.h

Go to the documentation of this file.

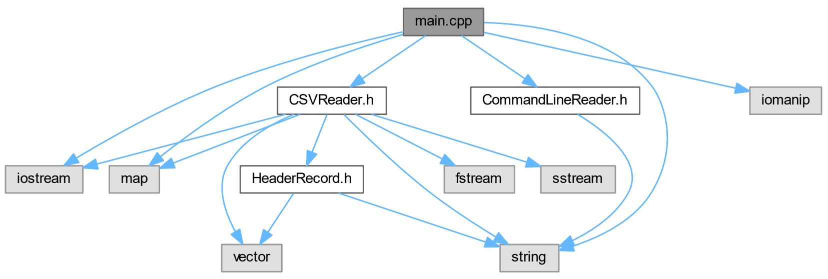
```

#ifndef ZIPCODES_HEADERRECORDBUFFER_H
#define ZIPCODES_HEADERRECORDBUFFER_H

#include <string>
#include <fstream>
#include "HeaderRecord.h"

class HeaderRecordBuffer {
public:
    HeaderRecordBuffer();

    ~HeaderRecordBuffer();

    // Method to read the header record from a file
    bool ReadHeaderRecord(const std::string& filename);

    // Method to write a header record to a file
    bool WriteHeaderRecord( HeaderRecord& header,std::string& fields);

    // Accessor methods to get and set the header record
    HeaderRecord GetHeaderRecord() const;

    void SetHeaderRecord(const HeaderRecord& header);

private:
    HeaderRecord headerRecord;

    //Method to change ASCII to integer for header size for reading
    //Method changes binary to integer for header size for reading
    int lengthDecoder(std::string header);

    std::vector<std::string> parser(std::string s);

};


#endif //ZIPCODES_HEADERRECORDBUFFER_H
```

4.17 main.cpp File Reference

This program reads a CSV file containing postal code data, calculates state statistics, and displays the easternmost, westernmost, northernmost, and southernmost locations for each state. It also makes a CommandLineReader instance to check for zipcodes and if location is present.

```
#include <iostream>
#include <map>
#include <string>
#include <iomanip>
#include "CSVReader.h"
#include "CommandLineReader.h"
```

Include dependency graph for main.cpp:



Functions

- void analyzeCSV (CSVReader &csvReader)

    Analyzes and displays state statistics from a CSVReader object.
- int main ()

    Main function to process and display state statistics from a CSV file and check location using commandLine.

4.17.1 Detailed Description

This program reads a CSV file containing postal code data, calculates state statistics, and displays the easternmost, westernmost, northernmost, and southernmost locations for each state. It also makes a CommandLineReader instance to check for zipcodes and if location is present.

Author

Chakong Lor

(Comments by Roshan) (Testing done by Shishir and Abdi) (Doxygen documentation by Abdi)

See also

CCSVReader.h, CCSVReader.cpp, CommandLineReader.h, CommandLineReader.cpp for Class declaration, implementation, and Assumptions.

The program utilizes the CSVReader class to process the CSV file. The methods are run twice on two different csv's. One contains the rows ordered by zip code, smallest to largest, The other csv is ordered by location name alphabetically A-Z. The two running's are compared to ensure that their output is the same.

Definition in file main.cpp.

4.17.2 Function Documentation

4.17.2.1 analyzeCSV()

```
void analyzeCSV (
             CSVReader & csvReader )
```

Analyzes and displays state statistics from a CSVReader object.

Parameters

csvReader	The CSVReader object to analyze.

Precondition

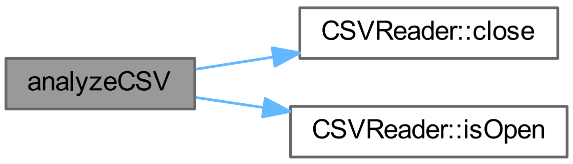
The CSVReader object is open and initialized.

Postcondition

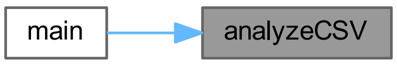
State statistics are displayed for the given CSVReader object.

Definition at line 63 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.17.2.2 main()

```
int main ( )
```

Main function to process and display state statistics from a CSV file and check location using commandLine.

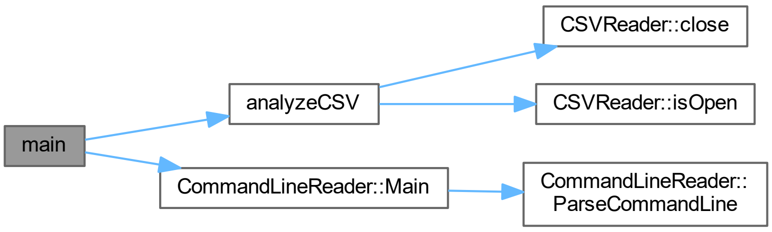
This function creates a CSVReader object, opens a CSV file, reads and processes the data, and displays state statistics. It also makes a CommandLineReader instance to check for zipcodes and if location is present

Returns

0 on success, 1 on failure (e.g., if the CSV file cannot be opened).

Definition at line 35 of file main.cpp.

Here is the call graph for this function:



4.18 main.cpp

Go to the documentation of this file.

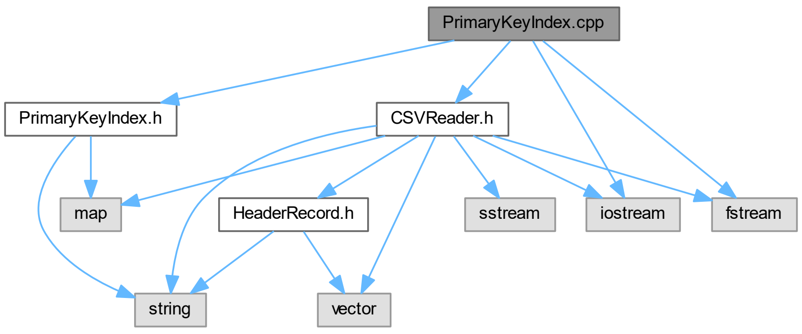
```
#include <iostream>
#include <map>
#include <string>
#include <iomanip>
#include "CSVReader.h"
#include "CommandLineReader.h"

// Declaration for analyzeCSV
void analyzeCSV(CSVReader &file);



int main() {
    //RunTest();
    // Create a CSVReader object and open a CSV file
    std::string file = "us_postal_codes.csv";
    std::cout « "Processing us_postal_codes.csv. \n" « std::endl;
    CSVReader csvReader(file);
    analyzeCSV(csvReader);

    std::string file2 = "us_postal_codes_place.csv";
    std::cout « "Processing us_postal_codes_ROWS_RANDOMIZED.csv. \n" « std::endl;
    CSVReader csvReader2(file2);
    analyzeCSV(csvReader2);

    std::cout « "\n" « std::endl;

    //check if location and its zipcode is in .csv file using commandline
    CommandLineReader cmdReader;
    cmdReader.Main();

    return 0;
}

void analyzeCSV(CSVReader &csvReader) {
    //CSVReader csvReader(fileName);
    if (!csvReader.isOpen()) {
        std::cerr « "Failed to open CSV file." « std::endl;
        return;
    }
    // Read and process the CSV file.
    csvReader.ReadFile();


    // Close the CSV file.
    csvReader.close();
}


```

4.19 PrimaryKeyIndex.cpp File Reference

Member function definitions for the PrimaryKeyIndex class.

```
#include "PrimaryKeyIndex.h"
#include "CSVReader.h"
#include <fstream>
#include <iostream>
```

Include dependency graph for PrimaryKeyIndex.cpp:



4.19.1 Detailed Description

Member function definitions for the PrimaryKeyIndex class.

Author

Fabian MullerDahlberg

See also

PrimaryKeyIndex.h for declaration.

Definition in file PrimaryKeyIndex.cpp.

4.20 PrimaryKeyIndex.cpp

Go to the documentation of this file.

```
#include "PrimaryKeyIndex.h"
#include "CSVReader.h"
#include <fstream>
#include <iostream>

PrimaryKeyIndex::PrimaryKeyIndex() {
    // Constructor: You can initialize any member variables here.
}

std::map<std::string, std::streampos> PrimaryKeyIndex::BuildIndex(std::string filename) {
    std::ifstream inputFile(filename, std::ios::binary);
    if (!inputFile) {
        std::cerr « "Failed to open input file." « std::endl;
        return std::map<std::string, std::streampos>();;
    }

    std::map<std::string, std::streampos> primaryKeyIndex;

    while (inputFile) {
        std::pair<std::size_t, std::string> record = CSVReader::ReadFromFile(inputFile);
        std::size_t size = record.first;
        std::string data = record.second;
        if (size == 0) {
            break; // End of file reached
        }

        std::vector<std::string> parsedRecord = CSVReader::ParseLine(data);

        if (!parsedRecord.empty()) {
            std::string key = parsedRecord[0];
            std::streampos currentRecordPos = inputFile.tellg();
            primaryKeyIndex[key] = currentRecordPos;
        }
    }
    return primaryKeyIndex;
}

std::map<std::string, std::streampos> PrimaryKeyIndex::ReadIndex(const std::string& fileName) {
    std::map<std::string, std::streampos> primaryKeyIndex;

    // Open the file for reading
    std::ifstream indexFile(fileName);
    if (indexFile.is_open()) {
        std::string line;
        while (std::getline(indexFile, line)) {
            std::string key;
            std::streampos value;

            // Split the line into key and value using a space
            size_t spacePos = line.find(' ');
            if (spacePos != std::string::npos) {
                key = line.substr(0, spacePos);
                value = std::stoll(line.substr(spacePos + 1));
                primaryKeyIndex[key] = value;
            }
        }
        // Close the file
        indexFile.close();
        std::cout « "Opened the index file for reading." « std::endl;
        return primaryKeyIndex;
    } else {
        std::cerr « "Error: Failed to open the index file for reading." « std::endl;
    }
}

void PrimaryKeyIndex::WriteIndex(const std::map<std::string, std::streampos> primaryKeyIndex) {
    // Implement the logic to write the primary key index to a file
    // Open the file for writing
    std::ofstream indexFile("KeyIndex.txt");
    if (indexFile.is_open()) {
        // Iterate through the map and write key-value pairs to the file
        for (const auto& pair : primaryKeyIndex) {
            indexFile « pair.first « " " « pair.second « std::endl;
        }
        // Close the file
        indexFile.close();
        std::cout « "Index written to index.txt" « std::endl;
    } else {
        std::cerr « "Error: Failed to open the index file for writing." « std::endl;
    }
}

bool PrimaryKeyIndex::SearchIndex(const std::map<std::string,std::string> recordIndex,std::string
filename) {
    std::ifstream inputFile(filename, std::ios::binary);
    // Implement the logic to search for a record in the index
    // Implement the logic to read the primary key index from a file
    std::string targetRecordId = "zipCode"; // The ID of the record you want to access
    //std::streampos targetPosition = recordIndex[targetRecordId];

// Move to the specified position in the file, accounting for the length indicator
    //inputFile.seekg(targetPosition + lengthIndicatorSize);

// Read the record at the specified position
// Read the length indicator
    std::size_t lengthIndicator;
    //inputFile.read(reinterpret_cast<char*>(&lengthIndicator), lengthIndicatorSize);

```

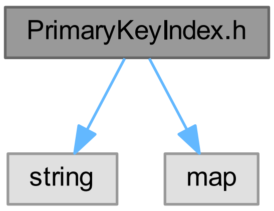
```
// Read the record data based on the length indicator
    std::string recordData(lengthIndicator, '\0');
    inputFile.read(&recordData[0], lengthIndicator);
    return false; // Return true if the record is found; false otherwise
}

void PrimaryKeyIndex::UnpackRecord(const std::string& recordData) {
    // Implement the logic to unpack and display a record
}
```

4.21 PrimaryKeyIndex.h File Reference

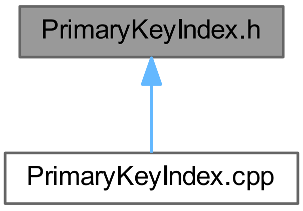
Declarations for class PrimaryKeyIndex.

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

Include dependency graph for PrimaryKeyIndex.h:



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



Classes

- class PrimaryKeyIndex

    Represents the Primary Key Index functionality. This class provides methods for building, reading, writing, searching, and unpacking a primary key index.

4.21.1 Detailed Description

Declarations for class PrimaryKeyIndex.

Author

Fabian MullerDahlberg

See also

PrimaryKeyIndex.cpp for the implementation of these functions.

This file declares the class PrimaryKeyIndex, which provides functionality to manage and operate on a primary key index. The class includes member functions for building, reading, writing, searching, and unpacking the primary key index.

Assumptions:

- The file used for indexing contains valid data.
- The primary key index uses a mapping between a string (as the key) and a stream position.
- Record data can be unpacked for display purposes using the provided methods.

Definition in file PrimaryKeyIndex.h.

4.22 PrimaryKeyIndex.h

Go to the documentation of this file.

```

#ifndef ZIPCODES_PRIMARYKEYINDEX_H
#define ZIPCODES_PRIMARYKEYINDEX_H

#include <string>
#include <map> // For using std::map or std::unordered_map

class PrimaryKeyIndex {
public:
    PrimaryKeyIndex();

    std::map<std::string, std::streampos> BuildIndex(std::string filename);

    std::map<std::string, std::streampos> ReadIndex(const std::string& fileName);

    void WriteIndex(const std::map<std::string, std::streampos> primaryKeyIndex);

    bool SearchIndex(const std::map<std::string, std::string> recordIndex, std::string filename);

    void UnpackRecord(const std::string& recordData);

private:
    // Define the data structure for the primary key index
    //std::map<std::string, std::streampos> primaryKeyIndex; // You can use an unordered_map if preferred
};

#endif //ZIPCODES_PRIMARYKEYINDEX_H
```

Index

~CSVReader
  CSVReader, 11
~HeaderRecordBuffer
  HeaderRecordBuffer, 25

analyzeCSV
  main.cpp, 60

BuildDataFile
  CSVReader, 11
buildFileStructure
  CSVReader, 11
BuildIndex
  PrimaryKeyIndex, 33

close
  CSVReader, 11
CommandLineReader, 5
  CommandLineReader, 6
  Main, 6
  ParseCommandLine, 7
  running, 8
CommandLineReader.cpp, 41, 42
CommandLineReader.h, 43, 45
ConvertToLength
  CSVReader, 12
county
  Row, 40
CSVReader, 9
  ~CSVReader, 11
  BuildDataFile, 11
  buildFileStructure, 11
  close, 11
  ConvertToLength, 12
  CSVReader, 10
  GenerateHeaderRecord, 12
  GetHeaders, 12
  Headers, 15
  isOpen, 13
  ParseLine, 13
  ReadFromFile, 14
  WriteToFile, 15
  ZipCSV, 15
CSVReader.cpp, 45, 47
CSVReader.h, 48, 50

fieldNames
  HeaderRecord, 22
fieldsPerRecord
  HeaderRecord, 22
fileName
  HeaderRecord, 22

GenerateHeaderRecord
  CSVReader, 12
getFieldsPerRecord
  HeaderRecord, 19
getFileName
  HeaderRecord, 19
GetHeaderRecord
  HeaderRecordBuffer, 26
GetHeaders
  CSVReader, 12
getHeaderSize
  HeaderRecord, 19
getPrimaryKeyIndexFileName
  HeaderRecord, 19
getPrimaryKeyOrdinality
  HeaderRecord, 19
getRecordCount
  HeaderRecord, 19
getRecordSizeBytes
  HeaderRecord, 19
getSizeFormatType
  HeaderRecord, 19
getVersion
  HeaderRecord, 20

HeaderRecord, 16
  fieldNames, 22
  fieldsPerRecord, 22
  fileName, 22
  getFieldsPerRecord, 19
  getFileName, 19
  getHeaderSize, 19
  getPrimaryKeyIndexFileName, 19
  getPrimaryKeyOrdinality, 19
  getRecordCount, 19
  getRecordSizeBytes, 19
  getSizeFormatType, 19
  getVersion, 20
  HeaderRecord, 18
  headerSize, 22
  primaryKeyIndexFileName, 22
  primaryKeyOrdinality, 22
  recordCount, 22
  recordSizeBytes, 22
  setFieldsPerRecord, 20
  setFileName, 20
  setHeaderSize, 20

setPrimaryKeyIndexFileName, 20
setPrimaryKeyOrdinality, 20
setRecordCount, 21
setRecordSizeBytes, 21
setSizeFormatType, 21
setVersion, 21
sizeFormatType, 23
version, 23
headerRecord
HeaderRecordBuffer, 31
HeaderRecord.cpp, 51
HeaderRecord.h, 52, 53
HeaderRecordBuffer, 23
∼HeaderRecordBuffer, 25
GetHeaderRecord, 26
headerRecord, 31
HeaderRecordBuffer, 25
lengthDecoder, 27
parser, 28
ReadHeaderRecord, 29
SetHeaderRecord, 30
WriteHeaderRecord, 31
HeaderRecordBuffer.cpp, 54, 55
HeaderRecordBuffer.h, 57, 59
Headers
CSVReader, 15
headerSize
HeaderRecord, 22

isOpen
CSVReader, 13

latitude
Row, 40
lengthDecoder
HeaderRecordBuffer, 27
longitude
Row, 40

Main
CommandLineReader, 6
main
main.cpp, 61
main.cpp, 59, 62
analyzeCSV, 60
main, 61

name
Row, 40

ParseCommandLine
CommandLineReader, 7
ParseLine
CSVReader, 13
parser
HeaderRecordBuffer, 28
PrimaryKeyIndex, 32
BuildIndex, 33
PrimaryKeyIndex, 33
ReadIndex, 34
SearchIndex, 35
UnpackRecord, 36
WriteIndex, 36
PrimaryKeyIndex.cpp, 63
PrimaryKeyIndex.h, 65, 66
primaryKeyIndexFileName
HeaderRecord, 22
primaryKeyOrdinality
HeaderRecord, 22

ReadFromFile
CSVReader, 14
ReadHeaderRecord
HeaderRecordBuffer, 29
ReadIndex
PrimaryKeyIndex, 34
Record, 37
recordData, 38
recordLength, 38
recordCount
HeaderRecord, 22
recordData
Record, 38
recordLength
Record, 38
recordSizeBytes
HeaderRecord, 22
Row, 39
county, 40
latitude, 40
longitude, 40
name, 40
state, 40
zip, 40
running
CommandLineReader, 8

SearchIndex
PrimaryKeyIndex, 35
setFieldsPerRecord
HeaderRecord, 20
setFileName
HeaderRecord, 20
SetHeaderRecord
HeaderRecordBuffer, 30
setHeaderSize
HeaderRecord, 20
setPrimaryKeyIndexFileName
HeaderRecord, 20
setPrimaryKeyOrdinality
HeaderRecord, 20
setRecordCount
HeaderRecord, 21
setRecordSizeBytes
HeaderRecord, 21
setSizeFormatType
HeaderRecord, 21
setVersion

HeaderRecord, 21
sizeFormatType
HeaderRecord, 23
state
Row, 40

UnpackRecord
PrimaryKeyIndex, 36

version
HeaderRecord, 23

WriteHeaderRecord
HeaderRecordBuffer, 31
WriteIndex
PrimaryKeyIndex, 36
WriteToFile
CSVReader, 15

zip
Row, 40
ZipCSV
CSVReader, 15