Zip Code Group Project

Version 1.0

Generated by Doxygen 1.13.2

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

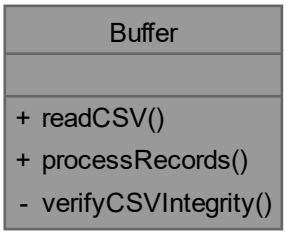
Class Documentation

3.1 Buffer Class Reference

A class to handle reading, processing, and validating ZIP code data.

`#include <Buffer.h>`

Collaboration diagram for Buffer:



Public Member Functions

- bool readCSV (const string &filename, vector< ZipCodeRecord > &records)

  Reads a CSV file, validates data integrity, and stores ZIP code records.
- void processRecords (const vector< ZipCodeRecord > &records, map< string, vector< ZipCodeRecord > > &state_map)

  Organizes ZIP code records by state.

Private Member Functions

- void verifyCSVIntegrity (const string &filename)

  Verifies the integrity of the CSV file and prints missing values.

3.1.1 Detailed Description

A class to handle reading, processing, and validating ZIP code data.

Definition at line 40 of file Buffer.h.

3.1.2 Member Function Documentation

3.1.2.1 readCSV()

```
bool Buffer::readCSV (
            const string & filename,
            vector< ZipCodeRecord > & records)
```

Reads a CSV file, validates data integrity, and stores ZIP code records.

Parameters

filename	The name of the CSV file to read.
records	Vector to store the read ZIP code records.

Returns

True if the file is read successfully, false otherwise.

Definition at line 19 of file Buffer.cpp.

3.1.2.2 processRecords()

```
void Buffer::processRecords (
            const vector< ZipCodeRecord > & records,
            map< string, vector< ZipCodeRecord > > & state_map)
```

Organizes ZIP code records by state.

Parameters

records	The vector of ZIP code records.
state_map	Map to store ZIP codes categorized by state.

Definition at line 92 of file Buffer.cpp.

3.1.2.3 verifyCSVIntegrity()

```
void Buffer::verifyCSVIntegrity (
            const string & filename)  [private]
```

Verifies the integrity of the CSV file and prints missing values.

Parameters

filename	The name of the CSV file to check.

Definition at line 102 of file Buffer.cpp.

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

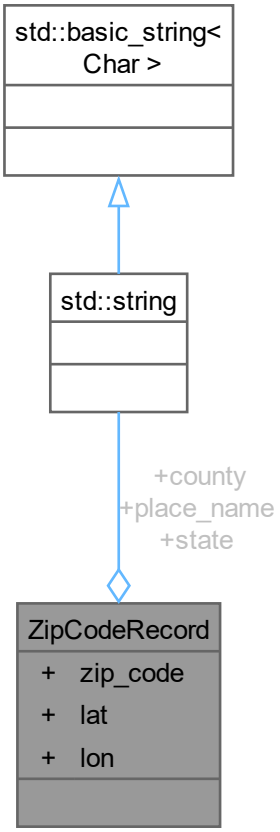
- Buffer.h
- Buffer.cpp

3.2 ZipCodeRecord Struct Reference

Structure to hold ZIP code data.

`#include <Buffer.h>`

Collaboration diagram for ZipCodeRecord:



Public Attributes

- int zip_code
  ZIP code.
- string place_name
  City or place name.
- string state
  Two-letter state abbreviation.
- string county
  County name.
- double lat
  Latitude.
- double lon
  Longitude.

3.2.1 Detailed Description

Structure to hold ZIP code data.

Definition at line 27 of file Buffer.h.

3.2.2 Member Data Documentation

3.2.2.1 zip_code

```
int ZipCodeRecord::zip_code
```

ZIP code.

Definition at line 28 of file Buffer.h.

3.2.2.2 place_name

```
string ZipCodeRecord::place_name
```

City or place name.

Definition at line 29 of file Buffer.h.

3.2.2.3 state

```
string ZipCodeRecord::state
```

Two-letter state abbreviation.

Definition at line 30 of file Buffer.h.

3.2.2.4 county

```
string ZipCodeRecord::county
```

County name.

Definition at line 31 of file Buffer.h.

3.2.2.5 lat

```
double ZipCodeRecord::lat
```

Latitude.

Definition at line 32 of file Buffer.h.

3.2.2.6 lon

```
double ZipCodeRecord::lon
```

Longitude.

Definition at line 33 of file Buffer.h.

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

- Buffer.h

Chapter 4

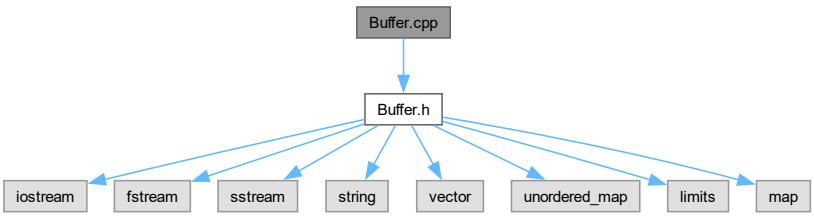
File Documentation

4.1 Buffer.cpp File Reference

Implementation of the Buffer class for handling ZIP code data processing and validation.

```
#include "Buffer.h"
```

Include dependency graph for Buffer.cpp:



4.1.1 Detailed Description

Implementation of the Buffer class for handling ZIP code data processing and validation.

This file contains the implementation of methods that read a CSV file, parse ZIP code records, validate their integrity, and categorize them by state.

Definition in file Buffer.cpp.

4.2 Buffer.cpp

Go to the documentation of this file.

```
00001
00008
00009 #include "Buffer.h"
00010
00011 using namespace std;
00012
00019 bool Buffer::readCSV(const string& filename, vector<ZipCodeRecord>& records) {
00020     ifstream file(filename);
00021     if (!file.is_open()) {
00022         cerr « "Error: Could not open the file " « filename « endl;
00023         return false;
00024     }
00025
00026     string line;
00027     getline(file, line); // Skip header
00028
00029     while (getline(file, line)) {
00030         stringstream ss(line);
00031         ZipCodeRecord record;
00032         string zip, lat, lon;
00033         vector<string> values;
00034         string token;
00035
00036         while (getline(ss, token, ',')) {
00037             values.push_back(token);
00038         }
00039
00040         // Ensure we have all expected columns
00041         if (values.size() != 6) {
00042             cerr « "Error: Incorrect number of columns on line: " « line « endl;
00043             continue; // Skip this entry, but do not terminate program
00044         }
00045
00046         // Extract values safely
00047         zip = values[0];
00048         record.place_name = values[1];
00049         record.state = values[2];
00050         record.county = values[3]; // County may be empty
00051         lat = values[4];
00052         lon = values[5];
00053
00054         // Print warnings for missing non-critical fields
00055         if (record.county.empty()) {
00056             cerr « "Warning: Missing county on line: " « line « endl;
00057         }
00058
00059         // Validate critical fields
00060         if (zip.empty() || record.state.empty() || lat.empty() || lon.empty()) {
00061             cerr « "Error: Missing critical values on line: " « line « endl;
00062             continue; // Skip entry, but keep processing
00063         }
00064
00065         // Ensure state is two characters
00066         if (record.state.length() != 2) {
00067             cerr « "Error: Invalid state format on line: " « line « endl;
00068             continue;
00069         }
00070
00071         try {
00072             record.zip_code = stoi(zip);
00073             record.lat = stod(lat);
00074             record.lon = stod(lon);
00075         } catch (const exception& e) {
00076             cerr « "Error parsing numeric values on line: " « line « " - " « e.what() « endl;
00077             continue;
00078         }
00079
00080         records.push_back(record);
00081     }
00082
00083     file.close();
00084     return true;
00085 }
00086
00092 void Buffer::processRecords(const vector<ZipCodeRecord>& records, map<string, vector<ZipCodeRecord»&
      state_map) {
00093     for (const auto& record : records) {
00094         state_map[record.state].push_back(record);
00095     }
00096 }
00097
00102 void Buffer::verifyCSVIntegrity(const string& filename) {
```

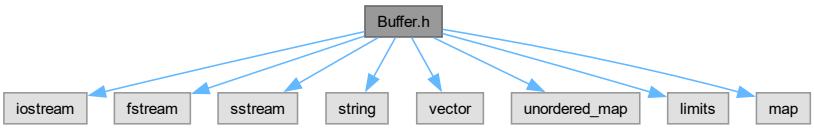
```
00103     ifstream file(filename);
00104     if (!file.is_open()) {
00105         cerr « "Error: Could not open the file " « filename « endl;
00106         return;
00107     }
00108
00109     string line;
00110     getline(file, line); // Skip header
00111
00112     while (getline(file, line)) {
00113         stringstream ss(line);
00114         vector<string> values;
00115         string token;
00116
00117         while (getline(ss, token, ',')) {
00118             values.push_back(token);
00119         }
00120
00121         // Ensure we have all expected columns
00122         if (values.size() != 6) {
00123             cerr « "Error: Incorrect number of columns on line: " « line « endl;
00124             continue; // Skip this entry, but do not terminate program
00125         }
00126
00127         // Validate critical fields
00128         if (values[0].empty() || values[2].empty() || values[4].empty() || values[5].empty()) {
00129             cerr « "Error: Missing critical values on line: " « line « endl;
00130         }
00131     }
00132
00133     file.close();
00134 }
```

4.3 Buffer.h File Reference

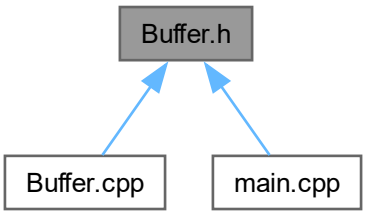
Header file for the Buffer class, which handles reading, processing, and validating ZIP code data.

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <unordered_map>
#include <limits>
#include <map>
```

Include dependency graph for Buffer.h:



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



Classes

- struct ZipCodeRecord

  Structure to hold ZIP code data.
- class Buffer

  A class to handle reading, processing, and validating ZIP code data.

4.3.1 Detailed Description

Header file for the Buffer class, which handles reading, processing, and validating ZIP code data.

This class reads a CSV file containing ZIP codes, organizes the data by state, and verifies data integrity before processing.

Definition in file Buffer.h.

4.4 Buffer.h

Go to the documentation of this file.

```
00001
00008
00009 #ifndef BUFFER_H
00010 #define BUFFER_H
00011
00012 #include <iostream>
00013 #include <fstream>
00014 #include <sstream>
00015 #include <string>
00016 #include <vector>
00017 #include <unordered_map>
00018 #include <limits>
00019 #include <map>
00020
00021 using namespace std;
00022
00027 struct ZipCodeRecord {
00028     int zip_code;
00029     string place_name;
00030     string state;
00031     string county;
00032     double lat;
00033     double lon;
00034 };
```

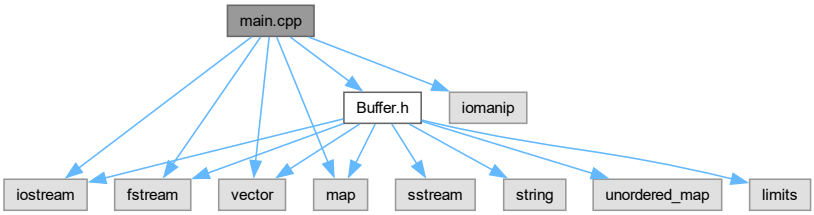
```
00035
00040 class Buffer {
00041 public:
00048     bool readCSV(const string& filename, vector<ZipCodeRecord>& records);
00049
00055     void processRecords(const vector<ZipCodeRecord>& records, map<string, vector<ZipCodeRecord>&
      state_map);
00056
00057 private:
00062     void verifyCSVIntegrity(const string& filename);
00063 };
00064
00065 #endif // BUFFER_H
```

4.5 main.cpp File Reference

Main program for processing ZIP code data and generating reports.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <map>
#include <iomanip>
#include "Buffer.h"
```

Include dependency graph for main.cpp:



Functions

- int main ()

  Main function to process ZIP code data.

4.5.1 Detailed Description

Main program for processing ZIP code data and generating reports.

This program reads ZIP code data from a CSV file (default: us_postal_codes.csv), but allows the user to enter a different file name if desired. It organizes the data by state and outputs the easternmost, westernmost, northernmost, and southernmost ZIP codes for each state to a text file and a CSV file.

Definition in file main.cpp.

4.5.2 Function Documentation

4.5.2.1 main()

```
int main ()
```

Main function to process ZIP code data.

The program first prompts the user to either use the default input file or enter a custom file name. It then reads ZIP code records from the chosen file, processes the records by state, and outputs the results to text and CSV files.

Returns

0 on successful execution, -1 on error.

Definition at line 29 of file main.cpp.

4.6 main.cpp

Go to the documentation of this file.

```
00001
00010
00011 #include <iostream>
00012 #include <fstream>
00013 #include <vector>
00014 #include <map>
00015 #include <iomanip>
00016 #include "Buffer.h"
00017
00018 using namespace std;
00019
00029 int main() {
00030     vector<ZipCodeRecord> records;
00031     Buffer buffer;
00032     map<string, vector<ZipCodeRecord» state_map;
00033
00034     // Default file
00035     string filename = "us_postal_codes.csv";
00036
00037     // Ask user if they want to change input file
00038     cout « "Default file input: \"" « filename « "\"\n";
00039     cout « "Would you like to change this? (Y/N): ";
00040     char choice;
00041     cin » choice;
00042
00043     if (toupper(choice) == 'Y') {
00044         cout « "Enter the input file name: ";
00045         cin » filename;
00046     }
00047
00048     // Read the CSV file
00049     if (!buffer.readCSV(filename, records)) {
00050         cerr « "Error: Unable to read CSV file: " « filename « endl;
00051         return -1;
00052     }
00053
00054     // Process ZIP code records and organize them by state
00055     buffer.processRecords(records, state_map);
00056
00057     // Open output files
00058     ofstream outfile_txt("LocationSortedZips.txt");
00059     ofstream outfile_csv("LocationSortedZips.csv");
00060     if (!outfile_txt || !outfile_csv) {
00061         cerr « "Error: Unable to open output files." « endl;
00062         return -1;
00063     }
00064
00065     // Write headers for both files
00066     outfile_txt « left « setw(5) « "State" « " | "
00067                 « right « setw(12) « "Easternmost" « " | "
00068                 « setw(12) « "Westernmost" « " | "
```

```
00069                 « setw(12) « "Northernmost" « " | "
00070                 « setw(12) « "Southernmost" « " |\n";
00071     outfile_txt « "=================================================================\n";
00072
00073     outfile_csv « "State,Easternmost,Westernmost,Northernmost,Southernmost\n";
00074
00075     // Process each state and find extreme ZIP codes
00076     for (const auto& entry : state_map) {
00077         const string& state = entry.first;
00078         const vector<ZipCodeRecord>& zipRecords = entry.second;
00079
00080         int east, west, north, south;
00081         double minLon = numeric_limits<double>::max();
00082         double maxLon = numeric_limits<double>::lowest();
00083         double maxLat = numeric_limits<double>::lowest();
00084         double minLat = numeric_limits<double>::max();
00085
00086         for (const auto& record : zipRecords) {
00087             if (record.lon < minLon) { minLon = record.lon; east = record.zip_code; }
00088             if (record.lon > maxLon) { maxLon = record.lon; west = record.zip_code; }
00089             if (record.lat > maxLat) { maxLat = record.lat; north = record.zip_code; }
00090             if (record.lat < minLat) { minLat = record.lat; south = record.zip_code; }
00091         }
00092
00093         outfile_txt « left « setw(5) « state « " | "
00094                     « right « setw(12) « east « " | "
00095                     « setw(12) « west « " | "
00096                     « setw(12) « north « " | "
00097                     « setw(12) « south « " |\n";
00098
00099         outfile_csv « state « "," « east « "," « west « "," « north « "," « south « "\n";
00100     }
00101
00102     // Close files and finish execution
00103     outfile_txt.close();
00104     outfile_csv.close();
00105     cout « "Output written to LocationSortedZips.txt and LocationSortedZips.csv\n";
00106     return 0;
00107 }
```

Index