Sequence Set

1.0

Generated by Doxygen 1.8.16

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 Block Class Reference

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

Public Member Functions

- Block ()

  Default constructor.
- Block (unsigned long long _RBN)

  Relative Block Number constructor.
- Block (string[ ])

  Constructor with record numbers.
- Block (string)

  Constructor with record numbers.
- void write (string)
- int search (string pKey)

  Searches for record.
- Block ∗ getNextBlock ()

  Gets pointer of next block.
- Block ∗ getPreviousBlock ()

  Gets pointer of previous block.
- void setNextBlock (Block ∗nextBlockPtr)

  Sets pointer to next block.
- void setPrevBlock (Block ∗previousBlockPtr)

  Sets pointer to previous block.
- int getRecordCount ()

  Gets the record count.
- int getLastRecordPKey ()

  Gets the last record of the block.
- bool deleteRecord (string pKey)
- bool addRecord (string pKey)
- void getRecords (Record block[ ])

  Gets zip codes for the sequence set.
- string blockData ()

  Returns RBN and records of the block.
- unsigned long long getRBN ()
- void setRBN (unsigned long long)

3.1.1 Detailed Description

Definition at line 30 of file Block.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Block() [1/4]

```
Block::Block ( )
```

Default constructor.

Precondition

None

Postcondition

A blank Block object is created

Definition at line 40 of file Block.cpp.

3.1.2.2 Block() [2/4]

```
Block::Block (
             unsigned long long _RBN )
```

Relative Block Number constructor.

Precondition

None

Postcondition

A blank Block object is created

Definition at line 55 of file Block.cpp.

3.1.2.3 Block() [3/4]

```
Block::Block (
            string [] )
```

Constructor with record numbers.

Precondition

The passed array must be of size fill count

Postcondition

A block object is made using an array of primary keys

3.1.2.4 Block() [4/4]

```
Block::Block (
            string _blockData )
```

Constructor with record numbers.

Precondition

A string

Postcondition

A Block object is created using the string

Definition at line 80 of file Block.cpp.

3.1.3 Member Function Documentation

3.1.3.1 addRecord()

```
bool Block::addRecord (
             string pKey )
```

Precondition

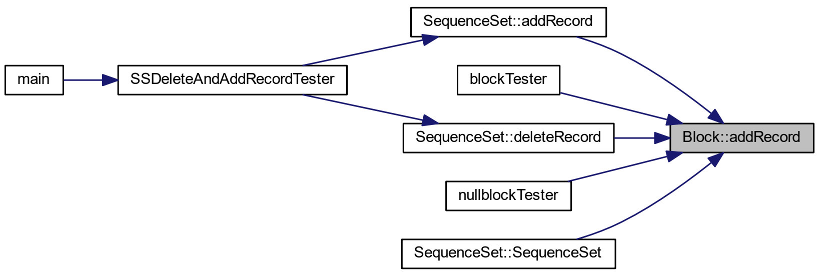
Primary key

Postcondition

Adds the record with the given primary key

Definition at line 260 of file Block.cpp.

Here is the caller graph for this function:



3.1.3.2 blockData()

```
string Block::blockData ( )
```

Returns RBN and records of the block.

Definition at line 70 of file Block.cpp.

3.1.3.3 deleteRecord()

```
bool Block::deleteRecord (
             string pKey )
```

Precondition

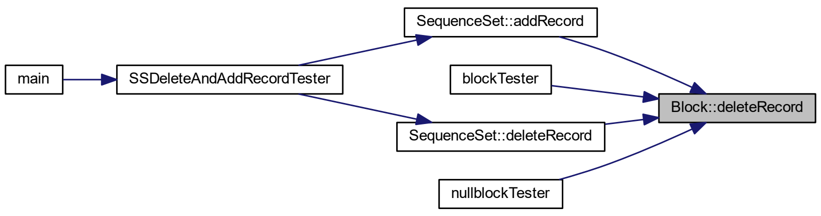
Primary key

Postcondition

Deletes the record with the given primary key

Definition at line 231 of file Block.cpp.

Here is the caller graph for this function:



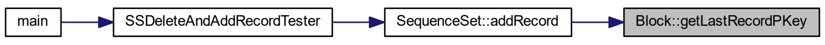
3.1.3.4 getLastRecordPKey()

```
int Block::getLastRecordPKey ( )
```

Gets the last record of the block.

Definition at line 225 of file Block.cpp.

Here is the caller graph for this function:



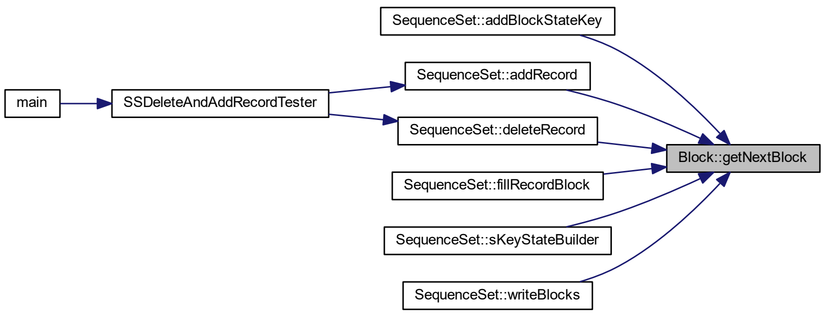
3.1.3.5 getNextBlock()

```
Block * Block::getNextBlock ( )
```

Gets pointer of next block.

Definition at line 192 of file Block.cpp.

Here is the caller graph for this function:


main
SSDeleteAndAddRecordTester
SequenceSet::addBlockStateKey
SequenceSet::addRecord
SequenceSet::deleteRecord
SequenceSet::fillRecordBlock
SequenceSet::sKeyStateBuilder
SequenceSet::writeBlocks
Block::getNextBlock

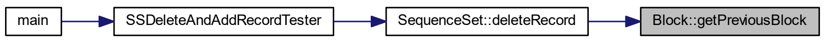
3.1.3.6 getPreviousBlock()

```
Block * Block::getPreviousBlock ( )
```

Gets pointer of previous block.

Definition at line 199 of file Block.cpp.

Here is the caller graph for this function:



3.1.3.7 getRBN()

```
unsigned long long Block::getRBN ( )
```

Definition at line 302 of file Block.cpp.

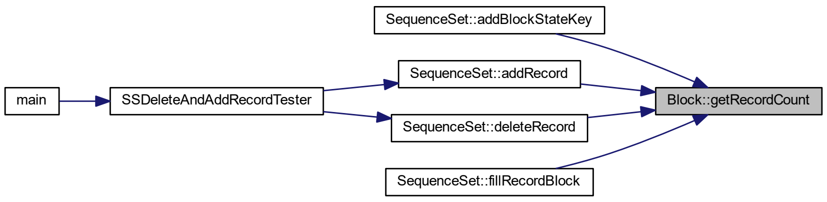
3.1.3.8 getRecordCount()

```
int Block::getRecordCount ( )
```

Gets the record count.

Definition at line 220 of file Block.cpp.

Here is the caller graph for this function:


SequenceSet::addBlockStateKey
SequenceSet::addRecord
main
SSDeleteAndAddRecordTester
Block::getRecordCount
SequenceSet::deleteRecord
SequenceSet::fillRecordBlock

3.1.3.9 getRecords()

```
void Block::getRecords (
             Record block[ ] )
```

Gets zip codes for the sequence set.

Precondition

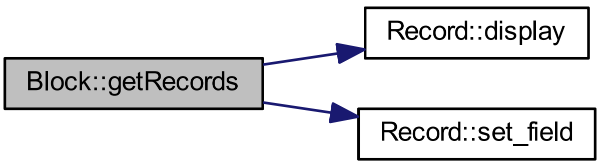
Block

Postcondition

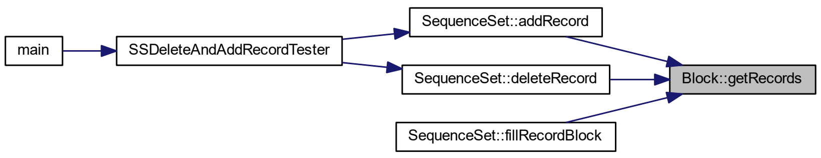
Returns records in a block

Definition at line 289 of file Block.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:


main
SSDeleteAndAddRecordTester
SequenceSet::addRecord
SequenceSet::deleteRecord
SequenceSet::fillRecordBlock
Block::getRecords

3.1.3.10 search()

```
int Block::search (
            string pKey )
```

Searches for record.

Precondition

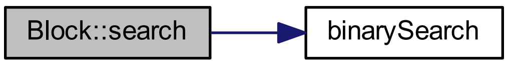
Primary key

Postcondition

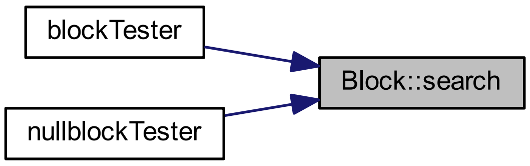
Returns the record or 0 if the record is not found Searches for primary key

Definition at line 185 of file Block.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



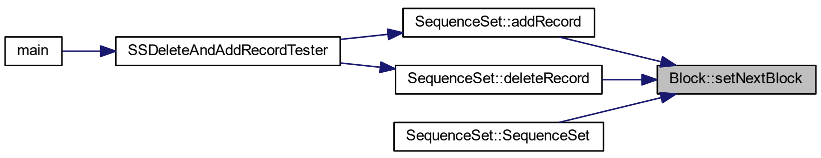
3.1.3.11 setNextBlock()

```
void Block::setNextBlock (
            Block * nextBlockPtr )
```

Sets pointer to next block.

Definition at line 206 of file Block.cpp.

Here is the caller graph for this function:



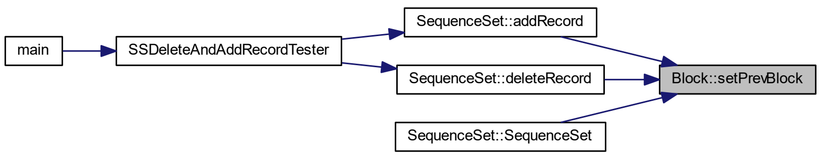
3.1.3.12 setPrevBlock()

```
void Block::setPrevBlock (
            Block * previousBlockPtr )
```

Sets pointer to previous block.

Definition at line 213 of file Block.cpp.

Here is the caller graph for this function:

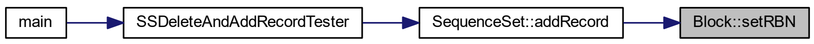


3.1.3.13 setRBN()

```
void Block::setRBN (
             unsigned long long RBN )
```

Definition at line 306 of file Block.cpp.

Here is the caller graph for this function:



3.1.3.14 write()

```
void Block::write (
             string _fileName )
```

Precondition

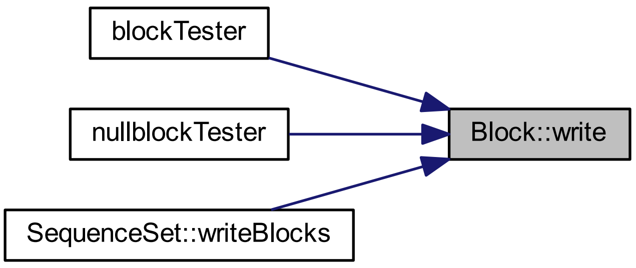
A block

Postcondition

Writes the block to a file

Definition at line 146 of file Block.cpp.

Here is the caller graph for this function:



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

- Doxygen/Input/Block.h
- Doxygen/Input/Block.cpp

3.2 Grid Class Reference

Grid class.

Public Member Functions

- Grid ()

    Default constructor.
- Grid (float, float)

    Constructor requiring both latitude and longitude.
- void setLatitude (float)

    Sets Latitude for this grid object.
- void setLongitude (float)

    Sets Longitude for this grid object.
- void setLatitude (string)

    Sets Latitude for this grid object.
- void setLongitude (string)

    Sets Longitude for this grid object.
- float getLatitude ()

    Gets Latitude for this grid object.
- float getLongitude ()

    Gets Longitude for this grid object.
- float getDistance (Grid)

    Gets Distance from this grid object to another grid object.

3.2.1 Detailed Description

Grid class.

Variables for latitude and longitude, constructor for setting 0 to both latitude and longitude (default constructor) and a constructor for setting latitude and longitude to input values.

Methods for setting and getting latitude and longitude and for getting the distance between two points.

Definition at line 30 of file grid.cpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Grid() `[1/2]`

```
Grid::Grid ( )
```

Default constructor.

Precondition

none

Postcondition

sets values for latitude and longitude to 0

Definition at line 51 of file grid.cpp.

3.2.2.2 Grid() [2/2]

```
Grid::Grid (
            float _latitude,
            float _longitude )
```

Constructor requiring both latitude and longitude.

Precondition

Values for latitude and longitude as float

Postcondition

Sets values for latitude and longitude

Definition at line 60 of file grid.cpp.

3.2.3 Member Function Documentation

3.2.3.1 getDistance()

```
float Grid::getDistance (
            Grid _grid )
```

Gets Distance from this grid object to another grid object.

Precondition

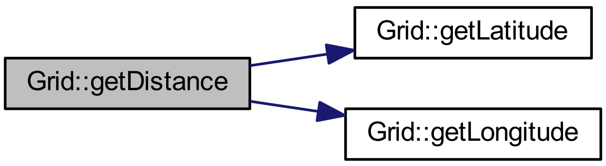
grid object must be provided

Postcondition

returns distance from this grid object to another grid object as float

Definition at line 117 of file grid.cpp.

Here is the call graph for this function:



3.2.3.2 getLatitude()

```
float Grid::getLatitude ( )
```

Gets Latitude for this grid object.

Precondition

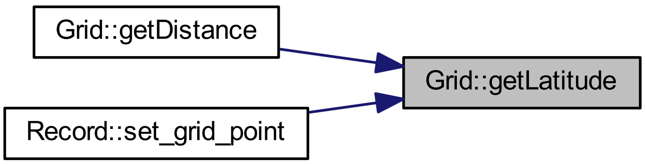
none

Postcondition

returns latitude for grid object as float

Definition at line 101 of file grid.cpp.

Here is the caller graph for this function:



3.2.3.3 getLongitude()

```
float Grid::getLongitude ( )
```

Gets Longitude for this grid object.

Precondition

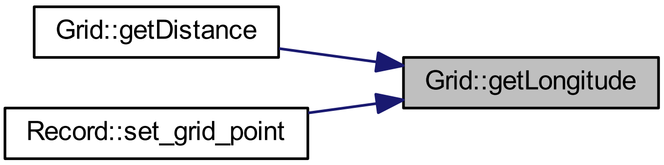
none

Postcondition

returns longitude for grid object as float

Definition at line 109 of file grid.cpp.

Here is the caller graph for this function:



3.2.3.4 setLatitude() [1/2]

```
void Grid::setLatitude (
            float _latitude )
```

Sets Latitude for this grid object.

Precondition

_latitude must follow rules regarding floats

Postcondition

Sets latitude for grid object

Definition at line 69 of file grid.cpp.

3.2.3.5 setLatitude() [2/2]

```
void Grid::setLatitude (
            string _latitude )
```

Sets Latitude for this grid object.

Precondition

_latitude must follow rules regarding string to float

Postcondition

Sets latitude for grid object

Definition at line 77 of file grid.cpp.

3.2.3.6 setLongitude() [1/2]

```
void Grid::setLongitude (
             float _longitude )
```

Sets Longitude for this grid object.

Precondition

_longitude must follow rules regarding floats

Postcondition

Sets longitude for grid object

Definition at line 85 of file grid.cpp.

3.2.3.7 setLongitude() [2/2]

```
void Grid::setLongitude (
             string _longitude )
```

Sets Longitude for this grid object.

Precondition

_longitude must follow rules regarding string to float

Postcondition

Sets longitude for grid object

Definition at line 93 of file grid.cpp.

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

- Doxygen/Input/grid.cpp

3.3 Record Class Reference

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

Public Member Functions

- Record ()

  Default constructor.
- Record (string, string, string, string, Grid)

  Constructor with a grid object.
- Record (string, string, string, string, string, string)

  Constructor that also takes latitude, and longitude.
- void display ()

  Displays all fields of the record.
- void display (string)

  Displays the specified field.
- string get_field (string)

  Get the desired field in the record to display a field from its data.
- void set_field (string, string)
- void set_longitude_latitude (float, float)

  Sets the latitude and longitude.
- void set_grid_point (Grid)

  Sets the Latitude and longitude based on a grid point.

3.3.1 Detailed Description

Definition at line 25 of file Record.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Record() `[1/3]`

```
Record::Record ( )
```

Default constructor.

Precondition

None

Postcondition

A blank record object is created

Definition at line 22 of file Record.cpp.

3.3.2.2 Record() [2/3]

```
Record::Record (
            string _zip_code,
            string _place_name,
            string _state,
            string _county,
            Grid _gridPoint )
```

Constructor with a grid object.

Precondition

Grid object is provided

Postcondition

A filled record object is created with a grid object

Definition at line 32 of file Record.cpp.

3.3.2.3 Record() [3/3]

```
Record::Record (
            string _zip_code,
            string _place_name,
            string _state,
            string _county,
            string latitude,
            string longitude )
```

Constructor that also takes latitude, and longitude.

Precondition

String is provided in order of latitude, longitude

Postcondition

A filled record object is created with a latitude and longitude

Definition at line 42 of file Record.cpp.

3.3.3 Member Function Documentation

3.3.3.1 display() [1/2]

```
void Record::display ( )
```

Displays all fields of the record.

Precondition

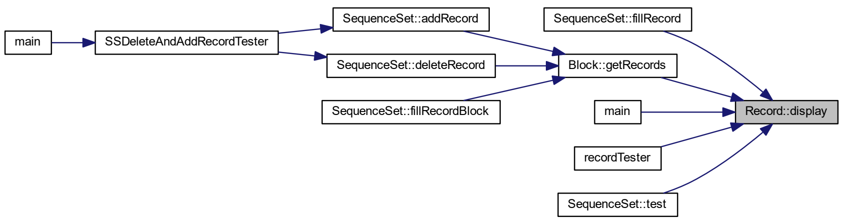
None

Postcondition

Record object will display all of its own data

Definition at line 71 of file Record.cpp.

Here is the caller graph for this function:



3.3.3.2 display() [2/2]

```
void Record::display (
            string field )
```

Displays the specified field.

Precondition

None

Postcondition

Record object will display specified field

Definition at line 84 of file Record.cpp.

3.3.3.3 get_field()

```
string Record::get_field (
             string field )
```

Get the desired field in the record to display a field from its data.

Precondition

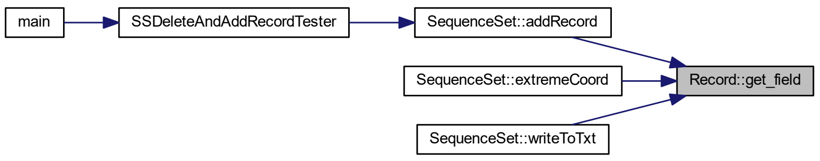
Provided string must match the name of a field in the record

Postcondition

Record object will display the specified field from its own data

Definition at line 109 of file Record.cpp.

Here is the caller graph for this function:



3.3.3.4 set_field()

```
void Record::set_field (
             string field,
             string data )
```

Precondition

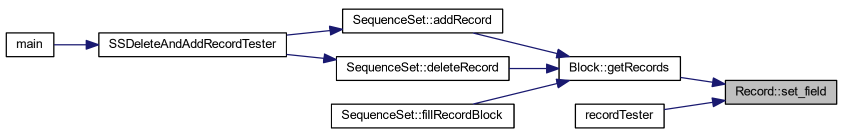
First provided string must match the name of a field in the record Second provided string must be the appropriate length for the field

Postcondition

Record object will display the specified field from its own data

Definition at line 137 of file Record.cpp.

Here is the caller graph for this function:


main
SSDeleteAndAddRecordTester
SequenceSet::addRecord
SequenceSet::deleteRecord
SequenceSet::fillRecordBlock
Block::getRecords
recordTester
Record::set_field

3.3.3.5 set_grid_point()

```
void Record::set_grid_point (
            Grid _gridPoint )
```

Sets the Latitude and longitude based on a grid point.

Precondition

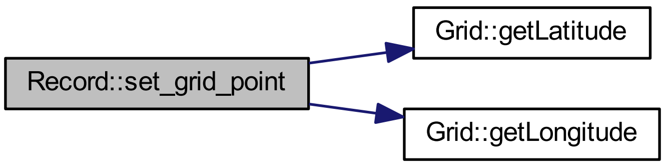
A grid point of type Grid

Postcondition

Sets latitude and longitude based on grid point recieved

Definition at line 172 of file Record.cpp.

Here is the call graph for this function:



3.3.3.6 set_longitude_latitude()

```
void Record::set_longitude_latitude (
            float longitude,
            float latitude )
```

Sets the latitude and longitude.

Precondition

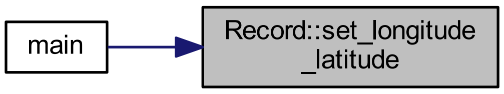
Provide longitude and latitude as floats

Postcondition

Set the latitude and longitude of the record

Definition at line 166 of file Record.cpp.

Here is the caller graph for this function:



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

- Doxygen/Input/Record.h
- Doxygen/Input/Record.cpp

3.4 SequenceSet Class Reference

```
#include <SequenceSet.h>
```

Public Member Functions

- SequenceSet ()
- void makeRecordOffsets (string fileName)

  Make record offsets.
- void fillIndex ()

  Fill Index.
- void fillRecordBlock (unsigned long long blockID)

  Fill record block.
- void writeBlocks ()

  Write blocks.
- Record fillRecord (string RecordString)

  Fill record.
- unsigned int getRecordCount ()

  Get record count.
- string fetch (string pKey)

  Fetch string.
- string fetch (unsigned int pKey)

  Fetch unsigned int.
- void addBlockStateKey (unsigned long long blockID)

  Add block state key.
- void sKeyStateBuilder ()

Add block state key builder.

- string extremeCoord (string, char)

  Extreme coordinate.

- int test ()

  Test 1 Preconditions: This is not a permanent function Postconditions: See precondition.

- bool deleteRecord (int pKey)
- void addRecord (Record record)
- void rewriteSSFile ()
- void writeToTxt (Record, string, string)

3.4.1 Detailed Description

Definition at line 27 of file SequenceSet.h.

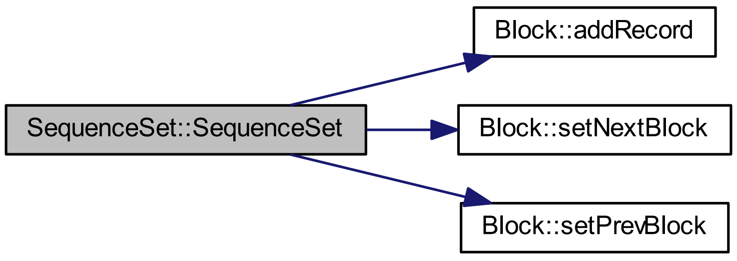
3.4.2 Constructor & Destructor Documentation

3.4.2.1 SequenceSet()

```
SequenceSet::SequenceSet ( )
```

Definition at line 30 of file SequenceSet.cpp.

Here is the call graph for this function:



3.4.3 Member Function Documentation

3.4.3.1 addBlockStateKey()

```
void SequenceSet::addBlockStateKey (
             unsigned long long blockID )
```

Add block state key.

Precondition

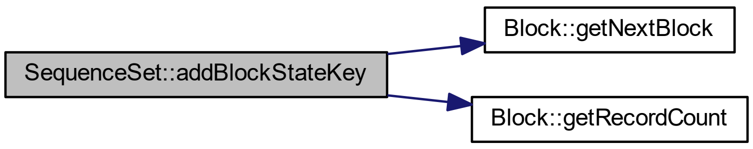
Block ID is less than block count

Postcondition

***add post

Definition at line 355 of file SequenceSet.cpp.

Here is the call graph for this function:

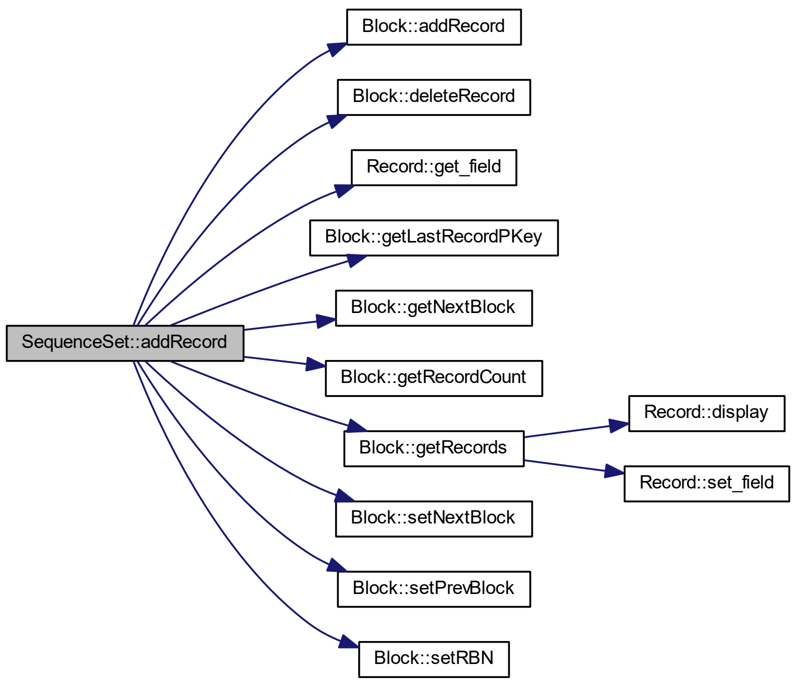


3.4.3.2 addRecord()

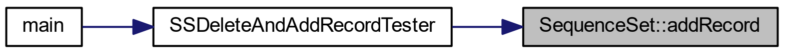
```
void SequenceSet::addRecord (
             Record record )
```

Definition at line 670 of file SequenceSet.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

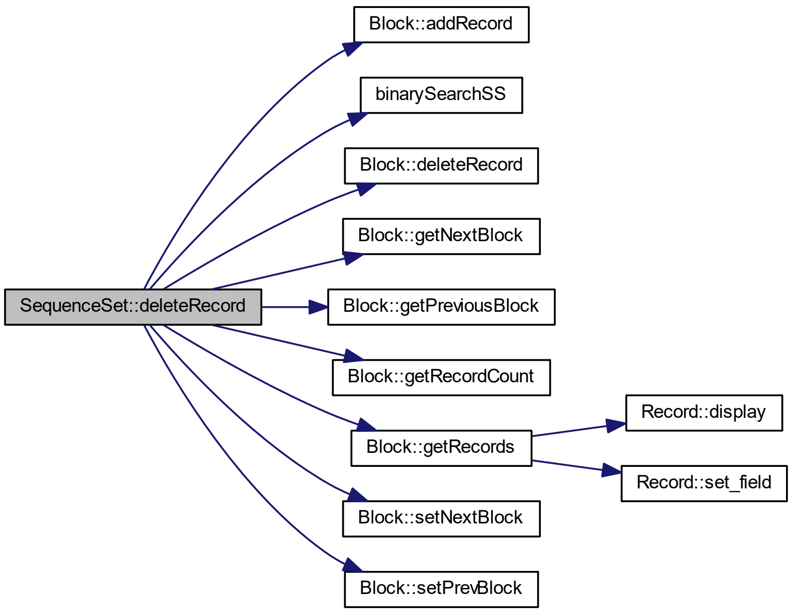


3.4.3.3 deleteRecord()

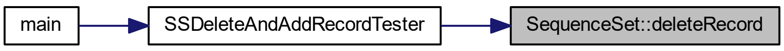
```
bool SequenceSet::deleteRecord (
            int pKey )
```

Definition at line 414 of file SequenceSet.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.4 extremeCoord()

```
string SequenceSet::extremeCoord (
             string state,
             char direction )
```

Extreme coordinate.

Precondition

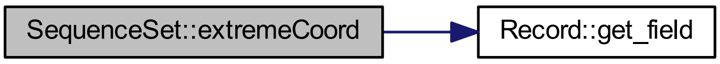
State of type string and Direction (N, E, S, W)

Postcondition

Returns the zipcode containing the most extreme point of said direction

Definition at line 519 of file SequenceSet.cpp.

Here is the call graph for this function:



3.4.3.5 fetch() [1/2]

```
string SequenceSet::fetch (
            string pKey )
```

Fetch string.

Precondition

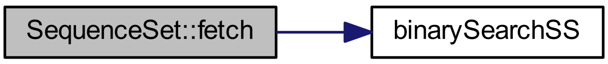
None

Postcondition

returns the whole record as a string

Definition at line 150 of file SequenceSet.cpp.

Here is the call graph for this function:



3.4.3.6 fetch() [2/2]

```
string SequenceSet::fetch (
             unsigned int pKey )
```

Fetch unsigned int.

Precondition

None

Postcondition

returns the whole record as a string

Definition at line 175 of file SequenceSet.cpp.

3.4.3.7 fillIndex()

```
void SequenceSet::fillIndex ( )
```

Fill Index.

Precondition

"RecordOffsets.txt" file must exist makeRecordOffsets can be ran to be sure of this

Postcondition

The index is made and stored here, in the Sequence Set

Definition at line 118 of file SequenceSet.cpp.

3.4.3.8 fillRecord()

```
Record SequenceSet::fillRecord (
             string RecordString )
```

Fill record.

Precondition

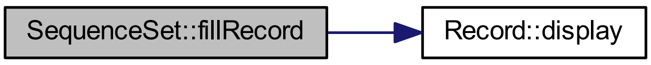
Record string must follow parameter conventions Record string must be complete, call fetch if needed

Postcondition

A record string is loaded into a record object

Definition at line 261 of file SequenceSet.cpp.

Here is the call graph for this function:



3.4.3.9 fillRecordBlock()

```
void SequenceSet::fillRecordBlock (
            unsigned long long blockID )
```

Fill record block.

Precondition

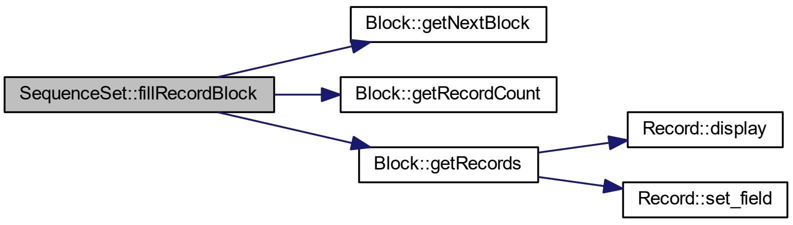
blockID must be less than the block count

Postcondition

Block is loaded into a record block

Definition at line 334 of file SequenceSet.cpp.

Here is the call graph for this function:



3.4.3.10 getRecordCount()

```
unsigned int SequenceSet::getRecordCount ( )
```

Get record count.

Precondition

Files must be available and the header in data file must contain "Records:"

Postcondition

RecordCount is returned

Definition at line 84 of file SequenceSet.cpp.

3.4.3.11 makeRecordOffsets()

```
void SequenceSet::makeRecordOffsets (
             string fileName )
```

Make record offsets.

Precondition

File must have fixed length primary keys equal to the "ziplength" in globals.cpp

Postcondition

An index file is made for the provided file name

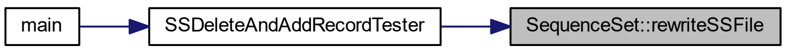
Definition at line 179 of file SequenceSet.cpp.

3.4.3.12 rewriteSSFile()

```
void SequenceSet::rewriteSSFile ( )
```

Definition at line 792 of file SequenceSet.cpp.

Here is the caller graph for this function:



3.4.3.13 sKeyStateBuilder()

```
void SequenceSet::sKeyStateBuilder ( )
```

Add block state key builder.

Precondition

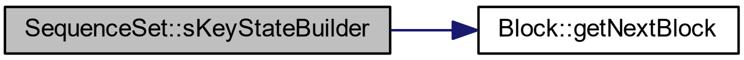
None

Postcondition

Builds the secondary index for states

Definition at line 659 of file SequenceSet.cpp.

Here is the call graph for this function:



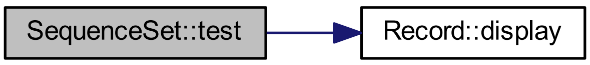
3.4.3.14 test()

```
int SequenceSet::test ( )
```

Test 1 Preconditions: This is not a permanent function Postconditions: See precondition.

Definition at line 609 of file SequenceSet.cpp.

Here is the call graph for this function:



3.4.3.15 writeBlocks()

```
void SequenceSet::writeBlocks ( )
```

Write blocks.

Precondition

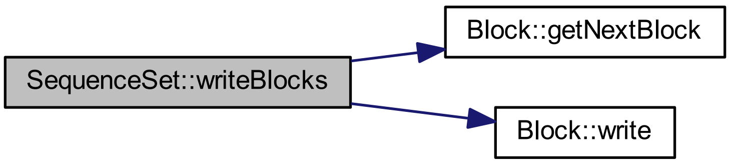
None

Postcondition

All blocks are called to run their write function

Definition at line 325 of file SequenceSet.cpp.

Here is the call graph for this function:

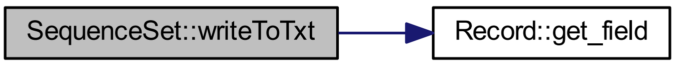


3.4.3.16 writeToTxt()

```
void SequenceSet::writeToTxt (
            Record record,
            string offset,
            string _fileName )
```

Definition at line 804 of file SequenceSet.cpp.

Here is the call graph for this function:



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

- Doxygen/Input/SequenceSet.h
- Doxygen/Input/SequenceSet.cpp

3.5 Truncate Class Reference

```
#include <Truncate.h>
```

Public Member Functions

- Truncate ()

  Default constructor Preconditions: None Postconditions: A truncate object will be created with a size of default for max length.
- Truncate (int)

  Default constructor Preconditions: Input must be an int Postconditions: A truncate object will be created with a size of the input for max length.
- string modifyString (string &)

  String modifier Preconditions: Input must be a string Postconditions: The truncate object will truncate the input string and return it.
- string truncatedString (string)

  Temporary string modifier Preconditions: Input must be a string Postconditions: The truncate object will copy the input string and return the truncated string without modifying the original.

3.5.1 Detailed Description

Definition at line 7 of file Truncate.h.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 Truncate() [1/2]

```
Truncate::Truncate ( )
```

Default constructor Preconditions: None Postconditions: A truncate object will be created with a size of default for max length.

Definition at line 10 of file Truncate.cpp.

3.5.2.2 Truncate() [2/2]

```
Truncate::Truncate (
             int _size )
```

Default constructor Preconditions: Input must be an int Postconditions: A truncate object will be created with a size of the input for max length.

Definition at line 16 of file Truncate.cpp.

3.5.3 Member Function Documentation

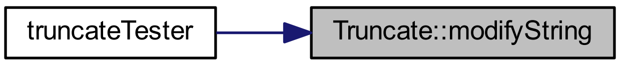
3.5.3.1 modifyString()

```
string Truncate::modifyString (
             string & _originalStr )
```

String modifier Preconditions: Input must be a string Postconditions: The truncate object will truncate the input string and return it.

Definition at line 27 of file Truncate.cpp.

Here is the caller graph for this function:



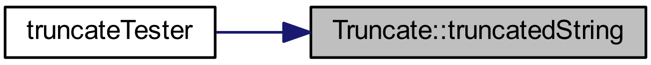
3.5.3.2 truncatedString()

```
string Truncate::truncatedString (
             string _string )
```

Temporary string modifier Preconditions: Input must be a string Postconditions: The truncate object will copy the input string and return the truncated string without modifying the original.

Definition at line 20 of file Truncate.cpp.

Here is the caller graph for this function:



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

- Doxygen/Input/Truncate.h
- Doxygen/Input/Truncate.cpp

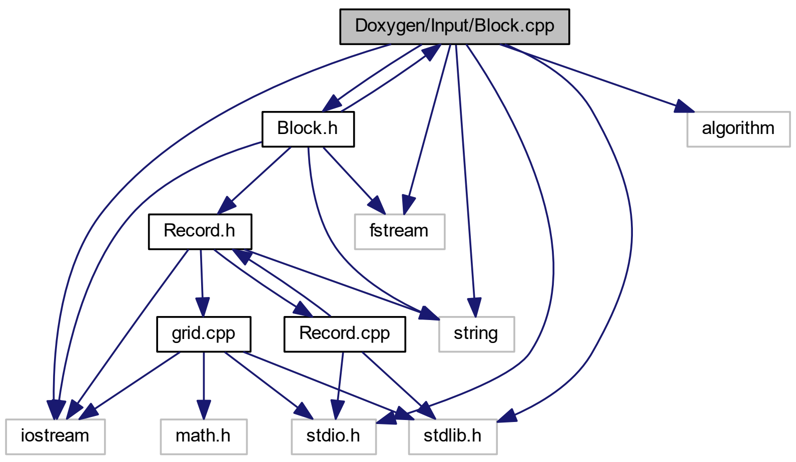
Chapter 4

File Documentation

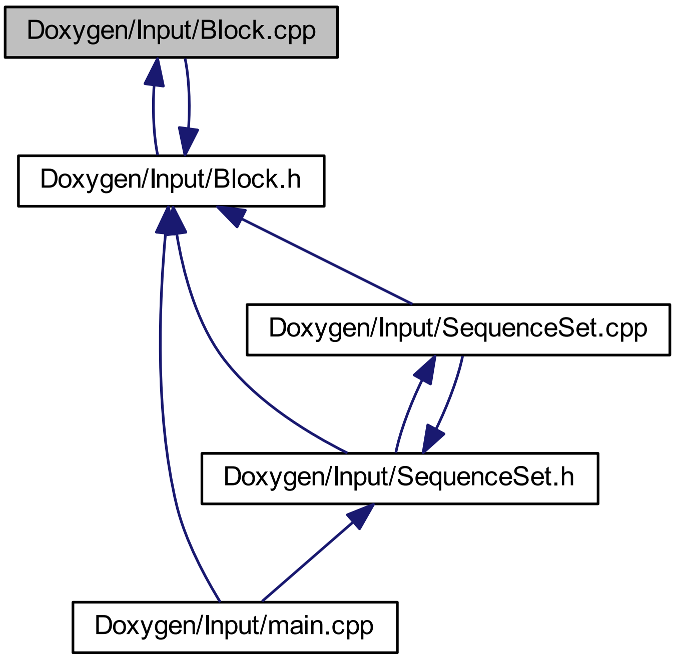
4.1 Doxygen/Input/Block.cpp File Reference

```
#include "Block.h"
#include <iostream>
#include <fstream>
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <algorithm>
```

Include dependency graph for Block.cpp:



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



Functions

- int binarySearch (const string arr[ ], string x, int n)

  Searches block for record by primary key.
- void convertStrArrToIntArr (const string strArr[ ], int intArr[ ], int ArrLength)

  String to integer.
- void convertIntArrToStrArr (string strArr[ ], int intArr[ ], int ArrLength)

  Integer to string.

Variables

- const string null_str = ""
- const int NULL_INT = 1000000

4.1.1 Function Documentation

4.1.1.1 binarySearch()

```
int binarySearch (
            const string arr[],
            string x,
            int n )
```

Searches block for record by primary key.

Precondition

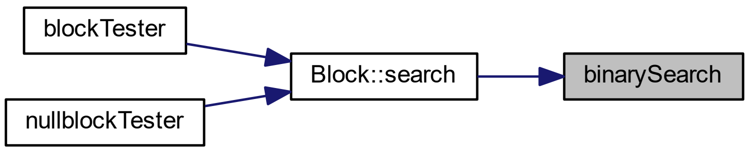
Primary key

Postcondition

Returns true if found otherwise returns false

Definition at line 328 of file Block.cpp.

Here is the caller graph for this function:



4.1.1.2 convertIntArrToStrArr()

```
void convertIntArrToStrArr (
            string strArr[],
            int intArr[],
            int ArrLength )
```

Integer to string.

Precondition

An array of integers

Postcondition

An array of strings

Definition at line 393 of file Block.cpp.

4.1.1.3 convertStrArrToIntArr()

```
void convertStrArrToIntArr (
            const string strArr[],
            int intArr[],
            int ArrLength )
```

String to integer.

Precondition

An array of strings

Postcondition

An array of integers

Definition at line 377 of file Block.cpp.

4.1.2 Variable Documentation

4.1.2.1 NULL_INT

```
const int NULL_INT = 1000000
```

Definition at line 38 of file Block.cpp.

4.1.2.2 null_str

```
const string null_str = ""
```

Definition at line 37 of file Block.cpp.

4.2 Block.cpp

```cpp

#include "Block.h"
#include <iostream>
#include <fstream>
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <algorithm>

using namespace std;

//prototype for binary search
int binarySearch(const string[], string,int);
void convertStrArrToIntArr(const string[], int[], int);
void convertIntArrToStrArr(string [], int [], int );

const string null_str = "";
const int NULL_INT = 1000000;

Block::Block()
{
  isEmpty = true;
  relativeBlockNumber = 0;
  recordCount = 0;
  for(int i = 0; i < RECORDSPERBLOCK; i++){
    records[i] = "";
  }

  nextBlock = nullptr;
  previousBlock = nullptr;

  if(DEBUG) {cout « "Made an empty block.\n";}
}

Block::Block(unsigned long long _RBN)
{
  isEmpty = true;
  relativeBlockNumber = _RBN;
  recordCount = 0;
  for(int i = 0; i < RECORDSPERBLOCK; i++){
    records[i] = "";
  }

  nextBlock = nullptr;
  previousBlock = nullptr;

  if(DEBUG) {cout « "Made an empty block.\n";}
}

string Block::blockData(){
  string returnString = "";
  returnString += relativeBlockNumber;
  for(int i = 0; i < recordCount; i++){
    returnString += " ";
    returnString += records[i];
  }
  return returnString;
}

Block::Block(string _blockData)
{
  if(DEBUG) {cout « "Making a block with \"" « _blockData « "\".\n";}

  isEmpty = false;
  relativeBlockNumber = 0;
  recordCount = 0;

  //set the primary keys of each record
  string tempStr = "";
  int recordNumber = 0;
  int j = 0; //pointer to track the position in the string
  while( j < _blockData.length() && j < BLOCKFILLCOUNT*ZIPLENGTH)
  {
    for(int i = 0; i < ZIPLENGTH; i++) //for each element of the pKey
    {
      if( _blockData[j] >= '0' && _blockData[j] <= '9' )
      {
        tempStr += _blockData[j]; //if the element is numeric, store the value
      }
      j++; //increment the pointer
    }
    records[recordNumber] = tempStr; //store the pKey in the class
    tempStr = ""; //clear the temp string
    if(records[recordNumber] != ""){
      recordCount++; //update the number of records in the block
```

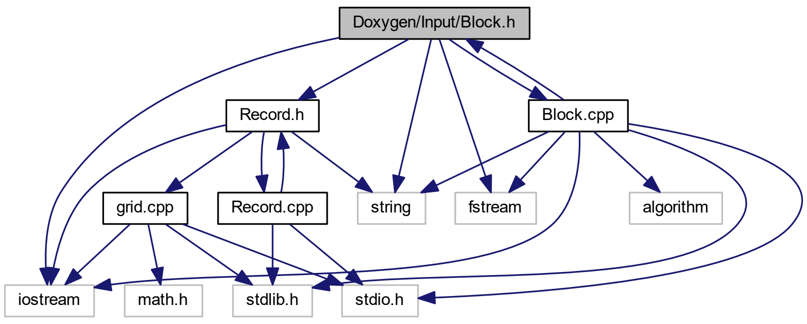
```
        recordNumber++; //increment the record number
      }
  }

  if(DEBUG) {cout « "Elements of Constructed block " « relativeBlockNumber « ": \"" ;
             for(int i = 0; i < RECORDSPERBLOCK; i++){cout « records[i] « " ";}
             cout « "\".\n";}

  nextBlock = nullptr;
  previousBlock = nullptr;
}

Block::Block(string _blockData[RECORDSPERBLOCK])
{
  if(DEBUG) {cout « "Making a block with \"" ;
             for(int i = 0; i < BLOCKFILLCOUNT; i++){cout « _blockData[i];}
             cout « "\".\n";}

  isEmpty = false;
  relativeBlockNumber = 0;
  recordCount = 0;

  //set the primary keys of each record
  for(int i = 0; i < BLOCKFILLCOUNT; i++)
  {
    records[i] = _blockData[i];
    if(records[i] != ""){
      recordCount++;
    }
  }

  if(DEBUG) {cout « "Elements of Constructed block " « relativeBlockNumber « ": \"" ;
             for(int i = 0; i < BLOCKFILLCOUNT; i++){cout « records[i] « " ";}
             cout « "\".\n";}

  nextBlock = nullptr;
  previousBlock = nullptr;
}


void Block::write(string _fileName)
{
  ofstream file;
  file.open(_fileName, ios_base::app);  if(DEBUG) {cout « "Writing block number " «
 relativeBlockNumber « " to a "« _fileName «".\n";}

  file.seekp(relativeBlockNumber * BLOCKLENGTH);

  if(DEBUG){
    cout « relativeBlockNumber « ": ";
    for(int i = 0; i < recordCount; i++){
      cout « records[i] « " ";
    }
    cout « endl;
  }

    file « "RBN " « relativeBlockNumber « ": ";
  if(DEBUG)(cout « "The file should read: \"");
    if(DEBUG){cout « "RBN " « relativeBlockNumber « ": ";}
    for(int i = 0;  i < recordCount; i++){
        string record = records[i];
        for(int j = ZIPLENGTH - record.length(); j > 0; j--){
            file « " ";
            if(DEBUG){cout « " ";}
        }

        file « record;
        if(DEBUG){cout « record;}
    }
  for(int i = RECORDSPERBLOCK - recordCount; i > 0; i--){
    for(int j = 0; j < ZIPLENGTH; j++){
      file « " ";
      if(DEBUG){cout « " ";}
    }
  }
  file « "\n";
  if(DEBUG)(cout « "\"\n");
  file.close();
}

int Block::search(string pKey)
{
    if(DEBUG) {cout « "Searching for " « pKey « " in this block\n";}

  return binarySearch(records, pKey, RECORDSPERBLOCK );
}

```

```
Block * Block::getNextBlock()
{
  return nextBlock;

    if(DEBUG) {cout « "Pointer to the next block has been returned.\n";}
}

Block * Block::getPreviousBlock()
{
  return previousBlock;

    if(DEBUG) {cout « "Pointer to the previous block has been returned.\n";}
}

void Block::setNextBlock( Block * nextBlockPtr )
{
  nextBlock = nextBlockPtr;

  if(DEBUG) {cout « "Pointer to the next block has been set.\n";}
}

void Block::setPrevBlock( Block * previousBlockPtr )
{
  previousBlock = previousBlockPtr;

  if(DEBUG) {cout « "Pointer to the previous block has been set.\n";}
}

int Block::getRecordCount()
{
  return recordCount;
}

int Block::getLastRecordPKey()
{
  if(DEBUG) {cout « "Getting last record of the block\n";}
  return stoi( records[ recordCount - 1 ] );
}

bool Block::deleteRecord(string pKey)
{
  if(DEBUG) {cout « "Deleting record " « pKey « " in block "« relativeBlockNumber «"\n";}
  if(DEBUG) {cout « "Elements of Constructed block before deleting record: \"" ;
      for(int i = 0; i < RECORDSPERBLOCK; i++){cout « records[i];}
      cout « "\".\n";}

  int position = this -> search(pKey); //get the position of the record to be deleted

  if ( position != -1 )
  {
    records[position] = ""; //delete the record
    recordCount--;  //decrement record count
    if(DEBUG) {cout « "Elements of Constructed block after deleting record: \"" ;
        for(int i = 0; i < RECORDSPERBLOCK; i++){if(records[i] == null_str){cout « "null";}else{cout « records[i];}}
        cout « "\".\n";}
    sortRecord(); //sort the record
    if(DEBUG) {cout « "Elements of Constructed block after sorting record: \"" ;
        for(int i = 0; i < RECORDSPERBLOCK; i++){if(records[i] == null_str){cout « "null";}else{cout « records[i];}}
        cout « "\".\n";}
    return true;
  }
    else
    {
      if(DEBUG) {cout « "Record not found in block. Could not delete"  « "\".\n";}
      return false;
    }
}

bool Block::addRecord(string pKey)
{
  if(DEBUG) {cout « "Adding a record to "« relativeBlockNumber «".\n";}
  if(DEBUG) {cout « "Elements of Constructed block before adding record: \"" ;
      for(int i = 0; i < RECORDSPERBLOCK; i++){cout « records[i];}
      cout « "\".\n";}

  for(int i = 0; i < RECORDSPERBLOCK; i++) //go through the block to see if there is empty record
  {
    if( records[i] == null_str) //if there is an empty record
    {
      records[i] = pKey;  //fill the record with the pKey
      recordCount++;  //increment record count
      if(DEBUG) {cout « "Elements of Constructed block after adding record: \"" ;
            for(int i = 0; i < RECORDSPERBLOCK; i++){cout « records[i];}
            cout « "\".\n";}
      sortRecord(); //sort the record
      if(DEBUG) {cout « "Elements of Constructed block after sorting record: \"" ;
            for(int i = 0; i < RECORDSPERBLOCK; i++){cout « records[i];}
            cout « "\".\n";}
      return true;
    }
  }

  if(DEBUG) {cout « "Block Full. Could not add record."  « "\".\n";}

  return false;
}

void Block::getRecords(Record block[])
{
  if(DEBUG) {cout « "Setting record zips\n";}

  for(auto i = 0; i < RECORDSPERBLOCK; i++){
    block[i].set_field("ZIP", records[i]);
    if(DEBUG){
      cout« "Block["«i«"]: " « endl;
      block[i].display();
    }
  }
}

unsigned long long Block::getRBN(){
  return relativeBlockNumber;
}

void Block::setRBN(unsigned long long RBN){
  relativeBlockNumber = RBN;
}

void Block::sortRecord()
{
  if(DEBUG) {cout « "Sorting the records in the block.\n";}

  int int_records_array[RECORDSPERBLOCK]; //to convert the string of records to integers
  convertStrArrToIntArr(records, int_records_array, RECORDSPERBLOCK);

  int n = sizeof(int_records_array)/sizeof(int_records_array[0]);
  sort(int_records_array, int_records_array+n);

  //convert back to strings and store in records array of string
  convertIntArrToStrArr(records, int_records_array, RECORDSPERBLOCK);
}

int binarySearch(const string arr[], string x, int n)
{
    int int_arr[n];
    int int_string;

    //convert the records (array of strings) to array of int
    for (int i = 0; i < n; i++)
    {
    if(arr[i] != null_str)
          int_arr[i] = stoi(arr[i]);
    }

    //convert string to find to int
    int_string = stoi(x);

    int l = 0 ;
    int r = n - 1;
    while (l <= r)
    {
      int m = l + (r - l) / 2;
          if(DEBUG) {cout « "mid: " « m «endl;}

      if(DEBUG) {cout « "comparing " « int_string « " and " « int_arr[m] «endl;}

        if ( int_arr[m] == int_string ){
        if(DEBUG) {cout « "record found" «endl;}
        return m;
          }

      // If x is greater, ignore left half
      if ( int_arr[m] < int_string ){
        l = m + 1;
      if(DEBUG) {cout « "new l: " « l «endl;}
      }

        // If x is smaller, ignore right half
      else{
        r = m - 1;
        if(DEBUG) {cout « "new r: " « l «endl;}
      }
    }

    return -1;
}

void convertStrArrToIntArr(const string strArr[], int intArr[], int ArrLength)
{
    //convert the records (array of strings) to array of int
    for (int i = 0; i < ArrLength; i++)
    {
    if(strArr[i] == null_str) //if the record is null
      intArr[i] = NULL_INT;
    else
      intArr[i] = stoi(strArr[i]);
    }
}

void convertIntArrToStrArr(string strArr[], int intArr[], int ArrLength)
{
  //convert the records (array of strings) to array of int
    for (int i = 0; i < ArrLength; i++)
    {
    if(intArr[i] == NULL_INT)//if the record is null
      strArr[i] = null_str;
    else
      strArr[i] = to_string(intArr[i]);
    }
}
```

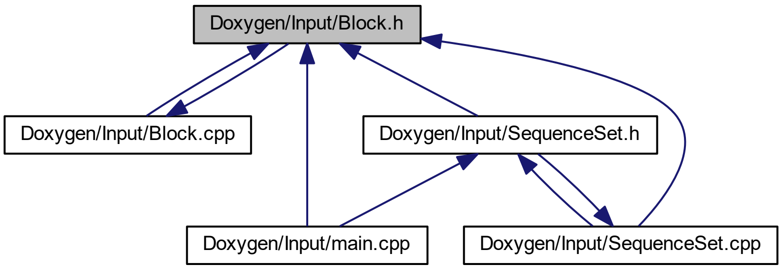
4.3 Doxygen/Input/Block.h File Reference

```
#include <iostream>
#include <string>
#include <fstream>
#include "Record.h"
#include "Block.cpp"
```

Include dependency graph for Block.h:



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



Classes

- class Block

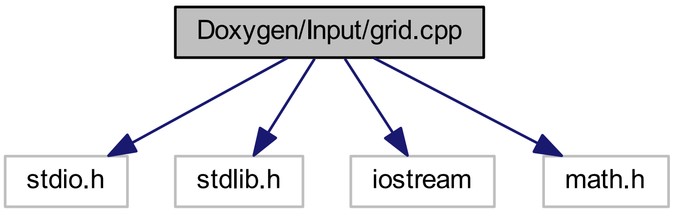
4.4 Block.h

```
#ifndef BLOCK_H
#define BLOCK_H

#include <iostream>
#include <string>
#include <fstream>
#include "Record.h"
using namespace std;

class Block
{
  public:
    Block();
    Block(unsigned long long _RBN);
        Block(string[]);
    Block(string);
        void write(string);
    int search(string pKey);
    Block * getNextBlock();
    Block * getPreviousBlock();
    void setNextBlock( Block * nextBlockPtr );
    void setPrevBlock( Block * previousBlockPtr );
    int getRecordCount();
    int getLastRecordPKey();
    bool deleteRecord(string pKey);
    bool addRecord(string pKey);
    void getRecords(Record block[]);
        string blockData();
    unsigned long long getRBN();
    void setRBN(unsigned long long);
  private:
    void sortRecord();
        bool isEmpty;
    unsigned long long relativeBlockNumber;
    int recordCount;
    string records[RECORDSPERBLOCK];
    Block * nextBlock;
    Block * previousBlock;
};

#include "Block.cpp"

#endif
```

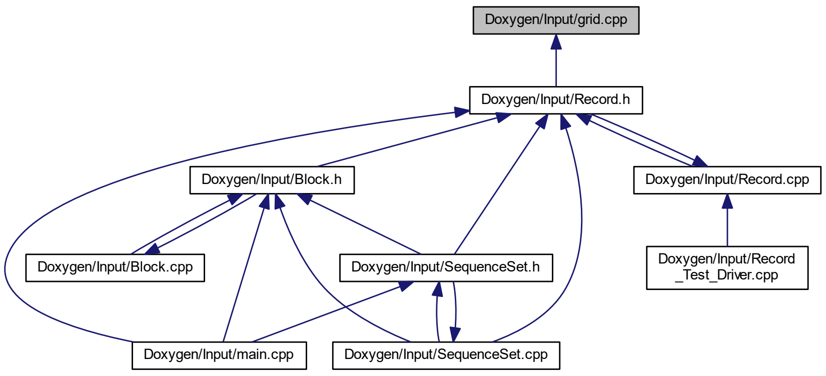
4.5 Doxygen/Input/grid.cpp File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <math.h>
```

Include dependency graph for grid.cpp:



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



Classes

- class Grid

  Grid class.

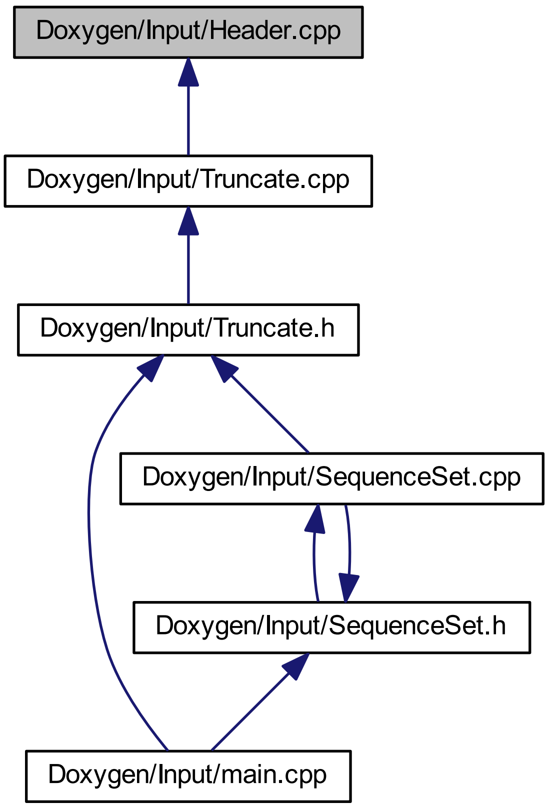
4.6 grid.cpp

```
00001
00014 #include <stdio.h>
00015 #include <stdlib.h>
00016 #include <iostream>
00017 #include <math.h>
00018 using namespace std;
```

```
//const bool DEBUG = true;

class Grid {
  private:
        float latitude;
        float longitude;
  public:
    Grid();
    Grid(float, float);
    void setLatitude(float);
    void setLongitude(float);
    void setLatitude(string);
    void setLongitude(string);
    float getLatitude();
        float getLongitude();
        float getDistance(Grid);
};

Grid::Grid(){
  latitude = 0;
  longitude = 0;
}

Grid::Grid(float _latitude, float _longitude){
    latitude = _latitude;
    longitude = _longitude;
}

void Grid::setLatitude(float _latitude){
    latitude = _latitude;
}

void Grid::setLatitude(string _latitude){
    setLatitude(stof(_latitude));
}

void Grid::setLongitude(float _longitude){
    longitude = _longitude;
}

void Grid::setLongitude(string _longitude){
    setLongitude(stof(_longitude));
}

float Grid::getLatitude(){
  return latitude;
}

float Grid::getLongitude(){
    return longitude;
}

float Grid::getDistance(Grid _grid){
  float distance = pow(latitude - _grid.getLatitude(),2) + pow(longitude - _grid.getLongitude(),2);
  distance = sqrt(distance);
  return distance;
}
```

4.7 Doxygen/Input/Header.cpp File Reference

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


Doxygen/Input/Header.cpp
Doxygen/Input/Truncate.cpp
Doxygen/Input/Truncate.h
Doxygen/Input/SequenceSet.cpp
Doxygen/Input/SequenceSet.h
Doxygen/Input/main.cpp

Variables

- const bool DEBUG = false

  Set true for debugging output

- const int RECORDSPERBLOCK = 4

  Maximum records for the block

- const int ZIPLENGTH = 6

  Max length of the zip code in digits.
- const int RBNLENGTH = 8

  Max length of the RBN code in digits.
- const int BLOCKLENGTH = RBNLENGTH + RECORDSPERBLOCK ∗ ZIPLENGTH

  Maximum length for the block

- const double FILLPERCENT = 75

  Max length of the RBN code in digits.

- const int BLOCKFILLCOUNT = RECORDSPERBLOCK ∗ (FILLPERCENT/100)

  Max length of the RBN code in digits.
- const string HEADERENDSTRING = "1234567890123456789012345678901234567890123456789012345678901234567890

  String at the end of the header.
- const string DATAFILENAME = "us_postal_codes.txt"

  Data file name.

4.7.1 Variable Documentation

4.7.1.1 BLOCKFILLCOUNT

```
const int BLOCKFILLCOUNT = RECORDSPERBLOCK * (FILLPERCENT/100)
```

Max length of the RBN code in digits.

4.7.1.2 BLOCKLENGTH

```
const int BLOCKLENGTH = RBNLENGTH + RECORDSPERBLOCK * ZIPLENGTH
```

Maximum length for the block

4.7.1.3 DATAFILENAME

```
const string DATAFILENAME = "us_postal_codes.txt"
```

Data file name.

4.7.1.4 DEBUG

```
const bool DEBUG = false
```

Set true for debugging output

4.7.1.5 FILLPERCENT

```
const double FILLPERCENT = 75
```

Max length of the RBN code in digits.

4.7.1.6 HEADERENDSTRING

```
const string HEADERENDSTRING = "123456789012345678901234567890123456789012345678901234567890123456789012345678
```

String at the end of the header.

4.7.1.7 RBNLENGTH

```
const int RBNLENGTH = 8
```

Max length of the RBN code in digits.

4.7.1.8 RECORDSPERBLOCK

```
const int RECORDSPERBLOCK = 4
```

Maximum records for the block

4.7.1.9 ZIPLENGTH

```
const int ZIPLENGTH = 6
```

Max length of the zip code in digits.

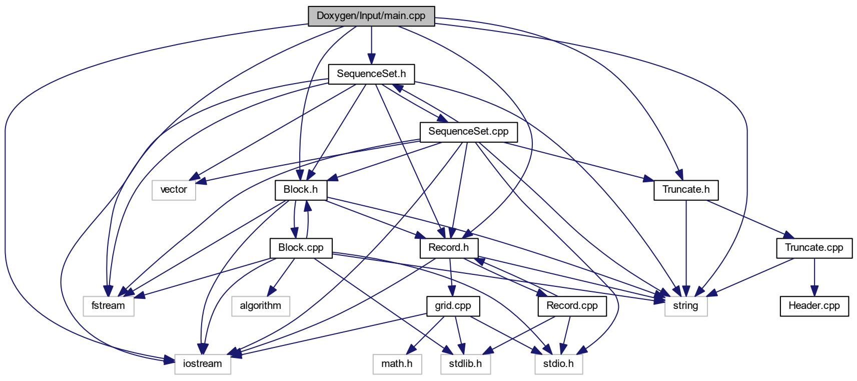
4.8 Header.cpp

```
extern const bool DEBUG                  = false;
extern const int RECORDSPERBLOCK = 4;
extern const int ZIPLENGTH           = 6;
extern const int RBNLENGTH          = 8;
extern const int BLOCKLENGTH          = RBNLENGTH + RECORDSPERBLOCK * ZIPLENGTH;
extern const double FILLPERCENT       = 75;
extern const int BLOCKFILLCOUNT       = RECORDSPERBLOCK * (FILLPERCENT/100);
extern const string HEADERENDSTRING =
      "1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890";

extern const string DATAFILENAME    = "us_postal_codes.txt";
```

4.9 Doxygen/Input/main.cpp File Reference

```
#include <iostream>
#include "Truncate.h"
#include "Record.h"
#include "Block.h"
#include "SequenceSet.h"
#include <string>
#include <fstream>
```

Include dependency graph for main.cpp:



Functions

- void truncateTester ()

  Tests the Truncate Class.
- void recordTester ()
- void blockTester ()
- void nullblockTester ()
- void SSDeleteAndAddRecordTester ()
- int main ()

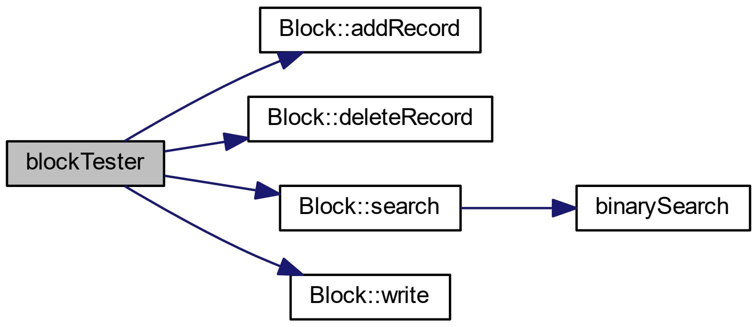
4.9.1 Function Documentation

4.9.1.1 blockTester()

```
void blockTester ( )
```

Definition at line 98 of file main.cpp.

Here is the call graph for this function:

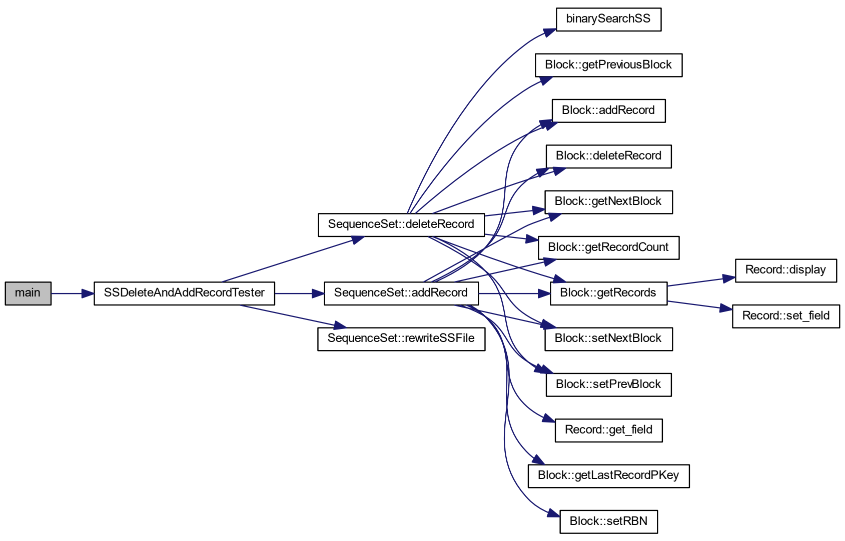


4.9.1.2 main()

```
int main ( )
```

Definition at line 17 of file main.cpp.

Here is the call graph for this function:

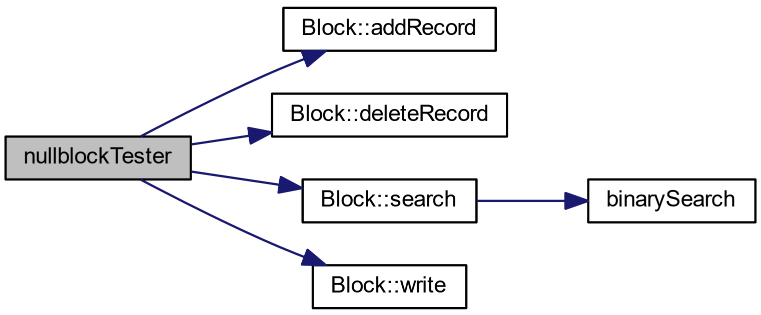


4.9.1.3 nullblockTester()

```
void nullblockTester ( )
```

Definition at line 65 of file main.cpp.

Here is the call graph for this function:

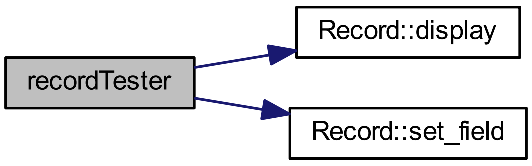


4.9.1.4 recordTester()

```
void recordTester ( )
```

Definition at line 141 of file main.cpp.

Here is the call graph for this function:

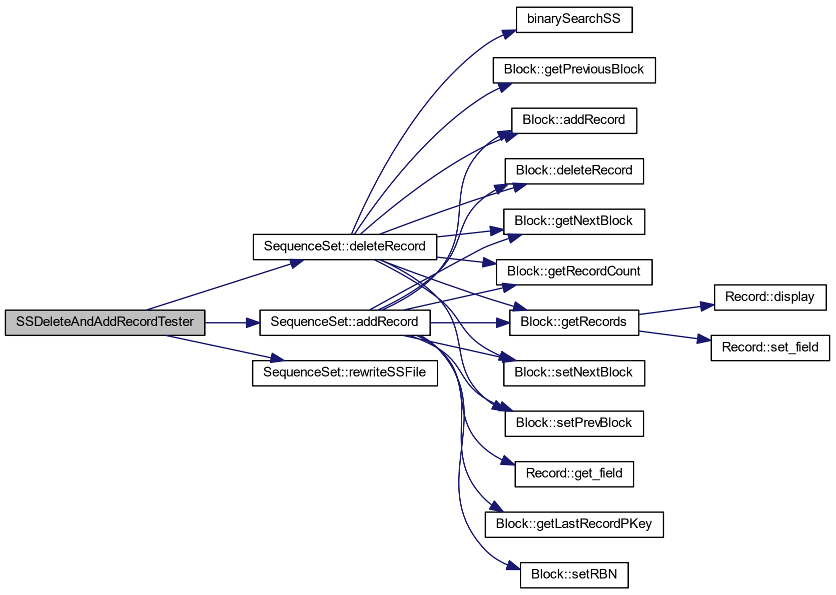


4.9.1.5 SSDeleteAndAddRecordTester()

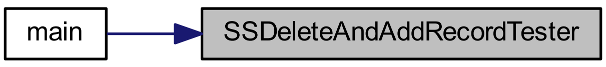
```
void SSDeleteAndAddRecordTester ( )
```

Definition at line 25 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



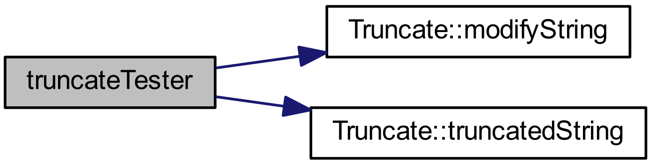
4.9.1.6 truncateTester()

```
void truncateTester ( )
```

Tests the Truncate Class.

Definition at line 130 of file main.cpp.

Here is the call graph for this function:



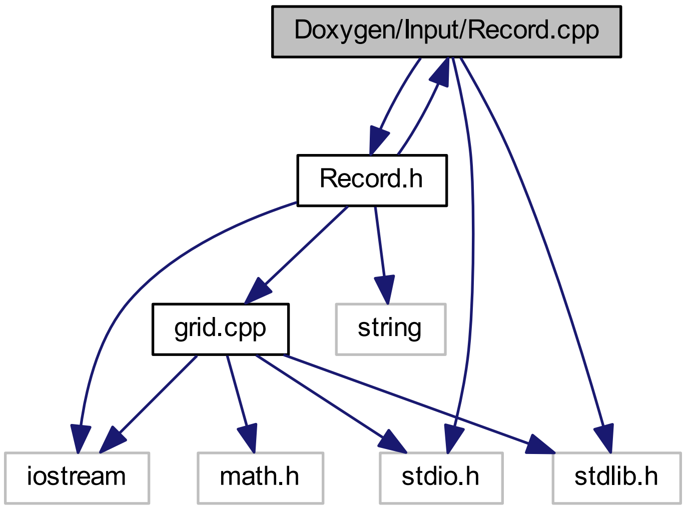
4.10 main.cpp

```
#include <iostream>
#include "Truncate.h"
#include "Record.h"
#include "Block.h"
#include "SequenceSet.h"
#include <string>
#include <fstream>

using namespace std;

void truncateTester();
void recordTester();
void blockTester();
void nullblockTester();
void SSDeleteAndAddRecordTester();

int main(){
  //nullblockTester();
  //recordTester();
    SSDeleteAndAddRecordTester();
  //SequenceSet SSClass; SSClass.test();
  return 0;
}

void SSDeleteAndAddRecordTester()
{
  SequenceSet SSClass;

  SSClass.deleteRecord(1008);
  SSClass.deleteRecord(1003);
  SSClass.deleteRecord(1004);

  string zip = "563";
  string place = "Little Falls";
  string state = "MN";
  string county = "Morrison";
  string longitude = "-74.25";
  string latitude = "79.72";
  Record testRecord(zip, place, state, county, longitude, latitude);
  SSClass.addRecord(testRecord);

  zip = "1024";
  Record testRecord2(zip, place, state, county, longitude, latitude);
  SSClass.addRecord(testRecord2);

  zip = "1025";
  Record testRecord3(zip, place, state, county, longitude, latitude);
  SSClass.addRecord(testRecord3);

  zip = "1051";
  Record testRecord4(zip, place, state, county, longitude, latitude);
  SSClass.addRecord(testRecord4);

  zip = "1052";
  Record testRecord5(zip, place, state, county, longitude, latitude);
  SSClass.addRecord(testRecord5);

  zip = "300";
  Record testRecord6(zip, place, state, county, longitude, latitude);
  SSClass.addRecord(testRecord6);

  SSClass.rewriteSSFile();
}

void nullblockTester(){
  Block * aBlock;
  ofstream sequenceSetFile;
  string fileName = "Sequence_Set.txt";
  sequenceSetFile.open(fileName);
  sequenceSetFile « "Hello File\n";
  sequenceSetFile.close();

  string records[4] = {"501", "544", "1001", ""};
  string blockInfo = "   501   544  1001  1002";

  //test block constructor
  Block anotherBlock(blockInfo);
  anotherBlock.write(fileName);

  //test block search method
  string recordTest = "1002";
  aBlock = new Block(1);
  cout « "Return 1 if the record was found: " « aBlock->search( recordTest ) « endl;

  recordTest = "103";
  aBlock->addRecord(recordTest);

  recordTest = "103";
  aBlock->addRecord(recordTest);

  recordTest = "544";
  aBlock->deleteRecord(recordTest);

  recordTest = "514";
  aBlock->deleteRecord(recordTest);
}

void blockTester(){
  Block aBlock;
  ofstream sequenceSetFile;
  string fileName = "Sequence_Set.txt";
  sequenceSetFile.open(fileName);
  sequenceSetFile « "Hello File\n";
  sequenceSetFile.close();

  string records[4] = {"501", "544", "1001", ""};
  string blockInfo = "   501   544  1001  1002";

  //test block constructor
  Block anotherBlock(blockInfo);
  anotherBlock.write(fileName);

  //test block search method
  string recordTest = "1002";
  cout « "Return 1 if the record was found: " « anotherBlock.search( recordTest ) « endl;

  recordTest = "103";
  anotherBlock.addRecord(recordTest);

  recordTest = "103";
  anotherBlock.addRecord(recordTest);

  recordTest = "544";
  anotherBlock.deleteRecord(recordTest);

  recordTest = "514";
  anotherBlock.deleteRecord(recordTest);
}

void truncateTester(){
  Truncate t;
  Truncate t2(5);
  string str = "123456789AB";

  cout « endl « "The String is " « str;
  cout « endl « "The String AS it is modified is " « t.modifyString(str);
  cout « endl « "The String IF it was modified is " « t2.truncatedString(str);
  cout « endl « "The String is " « str « endl;
}

void recordTester(){
  //test default constructor
    Record testRecord;
  cout « "Default constructor record (should be empty):";
  testRecord.display();
  cout « endl;

  //test fill record
  string zip = "56345";
  string place = "Little Falls";
  string state = "Minnesota";
  string county = "Morrison";
  string longitude = "-74.25";
  string latitude = "79.72";

  cout « "Fill Record with : " « zip « " " « place « " " « state « " " « county « " " « longitude « "
 " « latitude;

  testRecord.set_field( "z", zip );
  testRecord.set_field( "place", place );
  testRecord.set_field( "STATE", state );
  testRecord.set_field( "c", county );
  testRecord.set_field( "long", longitude );
  testRecord.set_field( "lat", latitude );

  testRecord.display();
  cout « endl;

  //test constructor 2
  float longitude_float = 74.25;
  float latitude_float = 79.72;

  Record testRecord2(zip, place, state, county, longitude, latitude);

  cout « "Constructor2 record (record should be full):";
  testRecord2.display();

    //test constructor 3
  Grid grid_test(longitude_float, latitude_float);

  Record testRecord3(zip, place, state, county, grid_test);

  cout « "Constructor3 record (record should be full):";
  testRecord3.display();

  //test display field
  cout « endl « "Test Display Field, display city:";
  testRecord3.display("CITY");
  cout « " expected: Little Falls" « endl;

  cout « "Test Display Field, display state:";
  testRecord3.display("STATE");
  cout « " expected: Minnesota" « endl;
}
```

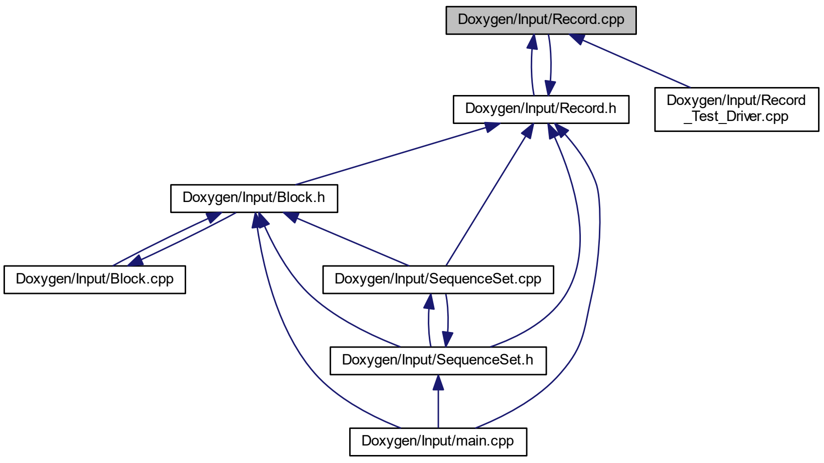
4.11 Doxygen/Input/Record.cpp File Reference

```
#include "Record.h"
#include <stdio.h>
#include <stdlib.h>
```

Include dependency graph for Record.cpp:



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



4.12 Record.cpp

```

#include "Record.h"
#include <stdio.h>
#include <stdlib.h>
using namespace std;

Record::Record()
{
  zip_code = "";
  place_name = "";
  state = "";
  county = "";
  this -> set_longitude_latitude( 0.0, 0.0 );
  if(DEBUG) {cout « "Made an empty record.\n";}
}

Record::Record(string _zip_code, string _place_name, string _state, string _county, Grid _gridPoint)
{
  zip_code = _zip_code;
  place_name = _place_name;
  state = _state;
  county = _county;
  this -> set_grid_point( _gridPoint );
  if(DEBUG) {cout « "Made a filled record using a gridPoint.\n";}
}

Record::Record(string _zip_code, string _place_name, string _state, string _county, string latitude, string longitude)
{
  float lon;
  float lat;

  try{
    lon = string_to_float( longitude );
  }
  catch(...){
    cout « "ERROR SETTING LONGITUDE, SETTING IT TO 0\n";
    lon = 0;
  }

  try{
    lat = string_to_float( latitude );
  }
  catch(...){
    cout « "ERROR SETTING LATITUDE IN " « zip_code « ", SETTING IT TO 0\n";
    lat = 0;
  }

  zip_code = _zip_code;
  place_name = _place_name;
  state = _state;
  county = _county;
  this -> set_longitude_latitude( lon, lat );
  if(DEBUG) {cout « "Made a filled record using string lat/longs.\n";}
}

void Record::display()
{
    if(DEBUG) {cout « "Displaying the whole record from the record.\n";}
    cout « endl
       « "Zipcode:\t" « get_field("Zip")
       « "\nPlace:\t\t" « get_field("City")
       « "\nState:\t\t" « get_field("State")
       « "\nCounty:\t\t" « get_field("County")
       « "\nLongitude:\t" « get_field("Longitude")
       « "\nLatitude:\t" « get_field("Latitude")
       « endl;
}

void Record::display(string field)
{
  if(DEBUG) {cout « "Displaying the "« field «" portion of the record.\t";}
  for(int i = 0; field[i] != NULL; i++){
      field[i] = toupper(field[i]);
  }

  if(field=="Z" || field=="ZIP")
    cout « zip_code « endl;
  else if(field=="CITY" || field=="P" || field=="PLACE_NAME")
    cout « place_name « endl;
  else if(field=="STATE")
    cout « state « endl;
  else if(field=="COUNTY")
    cout « county « endl;
  else if(field=="G" || field=="GRID")
    cout « gridPoint.getLatitude() « " " « gridPoint.getLongitude() « endl;
  else if(field == "LAT" || field == "LATITUDE")
    cout « gridPoint.getLatitude() « endl;
  else if(field == "LONG" || field == "LONGITUDE")
    cout « gridPoint.getLongitude() « endl;
  else
    cout « "Invalid field has been entered." « endl;
}

string Record::get_field(string field)
```

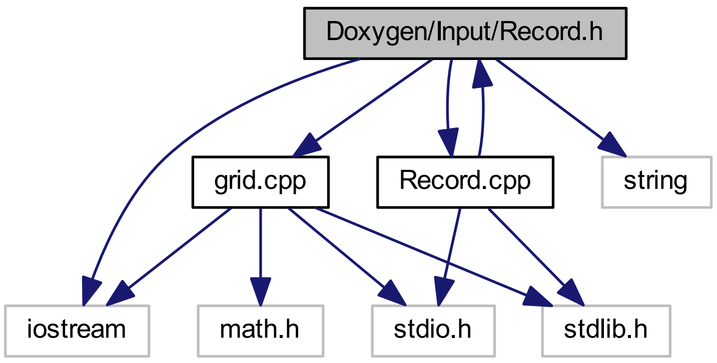
```
{
  if(DEBUG) {cout « "Retrieving the "« field «" portion of the record.\t";}
  string returnString;
  for(int i = 0; field[i] != NULL; i++){
      field[i] = toupper(field[i]);
  }

  if(field=="Z" || field=="ZIP")
    returnString = zip_code;
  else if(field=="CITY" || field=="P" || field=="PLACE_NAME")
    returnString = place_name;
  else if(field=="STATE")
    returnString = state;
  else if(field=="COUNTY")
    returnString = county;
  else if(field=="G" || field=="GRID")
    returnString = to_string(gridPoint.getLatitude()) + " " + to_string(gridPoint.getLongitude());
  else if(field == "LAT" || field == "LATITUDE")
    returnString = to_string(gridPoint.getLatitude());
  else if(field == "LONG" || field == "LONGITUDE")
    returnString = to_string(gridPoint.getLongitude());
  else
    returnString = "ERROR";

  return returnString;
}

void Record::set_field(string field, string data)
{
  if(DEBUG) {cout « "Setting the "« field «" portion of the record from "« get_field(field) « " to"«
 data «".\n";}
  for(int i = 0; field[i] != NULL; i++){
      field[i] = toupper(field[i]);
  }

  for(int i = 0; data[i] != NULL; i++){
      data[i] = toupper(data[i]);
  }

  if(field=="Z" || field=="ZIP")
    zip_code = data;
  else if(field=="CITY" || field=="P" || field=="PLACE_NAME")
    place_name = data;
  else if(field=="STATE")
    state = data;
  else if(field=="COUNTY")
    county = data;
  else if(field=="G" || field=="GRID")
    cout « "grid setter needs to implemented";
  else if(field == "LAT" || field == "LATITUDE")
    gridPoint.setLatitude(data);
  else if(field == "LONG" || field == "LONGITUDE")
    gridPoint.setLongitude(data);
  else
    cout « "ERROR" « endl;
}

void Record::set_longitude_latitude(float longitude, float latitude)
{
  gridPoint.setLatitude( latitude );
  gridPoint.setLongitude( longitude );
}

void Record::set_grid_point(Grid _gridPoint)
{
  gridPoint.setLatitude( _gridPoint.getLatitude() );
  gridPoint.setLongitude ( _gridPoint.getLongitude() );
}

//helper functions

float Record::string_to_float(string str)
{
  size_t size;
  float float_value = stof(str, &size);

  return float_value;
}









```

```

// /**-----------------------------------------------------------------------------
//  * @Record.cpp
//  * Class Record (Contains information about individual zipcodes)
//  * @author Tyler Lahr, Ryan Sweeney, and Seth Pomahatch
//  * (Additional comments by Mark Christenson)
//  *---------------------------------------------------------------------------
//  * Record class:  Used by Sequence Set Class
//  *   includes additional features:
//  *   -- Display the whole record it represents
//  *   -- Display a field with in the record
//  *   -- Return a field as a string
//  *   -- Return the latitude
//  *   -- Return the longitude
//  *---------------------------------------------------------------------------
//  */

// #include "Record.h"
// #include <stdio.h>
// #include <stdlib.h>
// using namespace std;

// Record::Record()
// {
//   zip_code = "";
//   place_name = "";
//   state = "";
//   county = "";
//   this -> set_longitude_latitude( 0.0, 0.0 );
//   if(DEBUG) {cout « "Made an empty record.\n";}
// }

// Record::Record(string _zip_code, string _place_name, string _state, string _county, Grid _gridPoint)
// {
//   zip_code = _zip_code;
//   place_name = _place_name;
//   state = _state;
//   county = _county;
//   this -> set_grid_point( _gridPoint );
//   if(DEBUG) {cout « "Made a filled record using a gridPoint.\n";}
// }

// Record::Record(string _zip_code, string _place_name, string _state, string _county, string latitude, string longitude)
// {
//   float lon = string_to_float( longitude );
//   float lat = string_to_float( latitude );

//   zip_code = _zip_code;
//   place_name = _place_name;
//   state = _state;
//   county = _county;
//   this -> set_longitude_latitude( lon, lat );
//   if(DEBUG) {cout « "Made a filled record using string lat/longs.\n";}
// }

// void Record::display()
// {
//   if(DEBUG) {cout « "Displaying the whole record from the record.\n";}
//   cout « endl
//       « "Zipcode: " « get_field("Zip")
//       « " Place: " « get_field("Place_name")
//        « " State: " « get_field("State")
//        « " County: " « get_field("County")
//        « " Longitude: " « get_field("Longitude")
//        « " Latitude: " « get_field("Latitude")
//        « endl;
// }

// void Record::display(string field)
// {
//   if(DEBUG) {cout « "Displaying the "« field «" portion of the record.\n";}
//   for(int i = 0; field[i] != NULL; i++){
//     field[i] = toupper(field[i]);
//   }

//   if(field=="Z" || field=="ZIP")
//     cout « zip_code « endl;
//   else if(field=="CITY" || field=="P" || field=="PLACE_NAME")
//     cout « place_name « endl;
//   else if(field=="STATE")
//     cout « state « endl;
//   else if(field=="COUNTY")
//     cout « county « endl;
//   else if(field=="G" || field=="GRID")
//     cout « gridPoint.getLatitude() « " " « gridPoint.getLongitude() « endl;
//   else if(field == "LAT" || field == "LATITUDE")
//     cout « gridPoint.getLatitude() « endl;
//   else if(field == "LONG" || field == "LONGITUDE")
//     cout « gridPoint.getLongitude() « endl;
//   else
//     cout « "Invalid field has been entered." « endl;
// }

// string Record::get_field(string field)
// {
//   if(DEBUG) {cout « "Retrieving the "« field «" portion of the record.\n";}
//   string returnString;
//   for(int i = 0; field[i] != NULL; i++){
//     field[i] = toupper(field[i]);
//   }

//   if(field=="Z" || field=="ZIP")
//     returnString = zip_code;
//   else if(field=="CITY" || field=="P" || field=="PLACE_NAME")
//     returnString = place_name;
//   else if(field=="STATE")
//     returnString = state;
//   else if(field=="COUNTY")
//     returnString = county;
//   else if(field=="G" || field=="GRID")
//     returnString = to_string(gridPoint.getLatitude()) + " " + to_string(gridPoint.getLongitude());
//   else if(field == "LAT" || field == "LATITUDE")
//     returnString = to_string(gridPoint.getLatitude());
//   else if(field == "LONG" || field == "LONGITUDE")
//     returnString = to_string(gridPoint.getLongitude());
//   else
//     returnString = "ERROR";

//   return returnString;
// }

// void Record::set_field(string field, string data)
// {
//   if(DEBUG) {cout « "Setting the "« field «" portion of the record from "« get_field(field) « "
   to"« data «".\n";}
//   for(int i = 0; field[i] != NULL; i++){
//       field[i] = toupper(field[i]);
//   }

//   for(int i = 0; data[i] != NULL; i++){
//       data[i] = toupper(data[i]);
//   }

//   if(field=="Z" || field=="ZIP")
//     zip_code = data;
//   else if(field=="CITY" || field=="P" || field=="PLACE_NAME")
//     place_name = data;
//   else if(field=="STATE")
//     state = data;
//   else if(field=="COUNTY")
//     county = data;
//   else if(field=="G" || field=="GRID")
//     cout « "grid setter needs to implemented";
//   else if(field == "LAT" || field == "LATITUDE")
//     gridPoint.setLatitude(data);
//   else if(field == "LONG" || field == "LONGITUDE")
//     gridPoint.setLongitude(data);
//   else
//     cout « "ERROR" « endl;
// }

// void Record::set_longitude_latitude(float longitude, float latitude)
// {
//   gridPoint.setLatitude( latitude );
//   gridPoint.setLongitude( longitude );
// }

// void Record::set_grid_point(Grid _gridPoint)
// {
//   gridPoint.setLatitude( _gridPoint.getLatitude() );
//   gridPoint.setLongitude ( _gridPoint.getLongitude() );
// }

// float Record::string_to_float(string str)
// {
//   size_t size;
//   float float_value = stof(str, &size);

//   return float_value;
// }

// //   if gridPoint.setLatitude(data);
// //   else if(field == "LONG" || field == "LONGITUDE")
// //     gridPoint.setLongitude(data);
// //   else
// //     cout « "ERROR" « endl;
// // }

// void Record::set_longitude_latitude(float longitude, float latitude)
// {
//   gridPoint.setLatitude( latitude );
//   gridPoint.setLongitude( longitude );
// }

// void Record::set_grid_point(Grid _gridPoint)
// {
//   gridPoint.setLatitude( _gridPoint.getLatitude() );
//   gridPoint.setLongitude ( _gridPoint.getLongitude() );
// }

// float Record::string_to_float(string str)
// {
//   size_t size;
//   float float_value = stof(str, &size);

//   return float_value;
// }
```

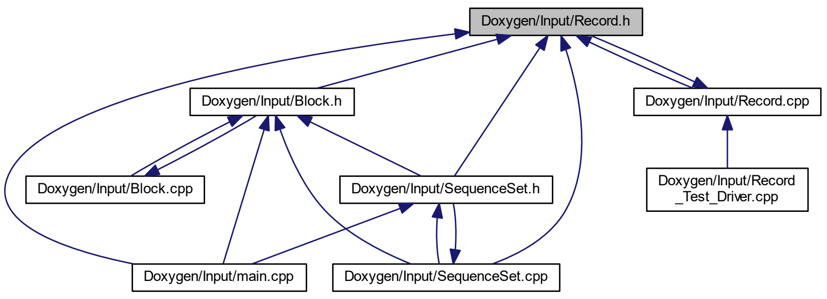
4.13 Doxygen/Input/Record.h File Reference

```
#include <iostream>
#include <string>
#include "grid.cpp"
#include "Record.cpp"
```

Include dependency graph for Record.h:



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



Classes

- class Record

4.14 Record.h

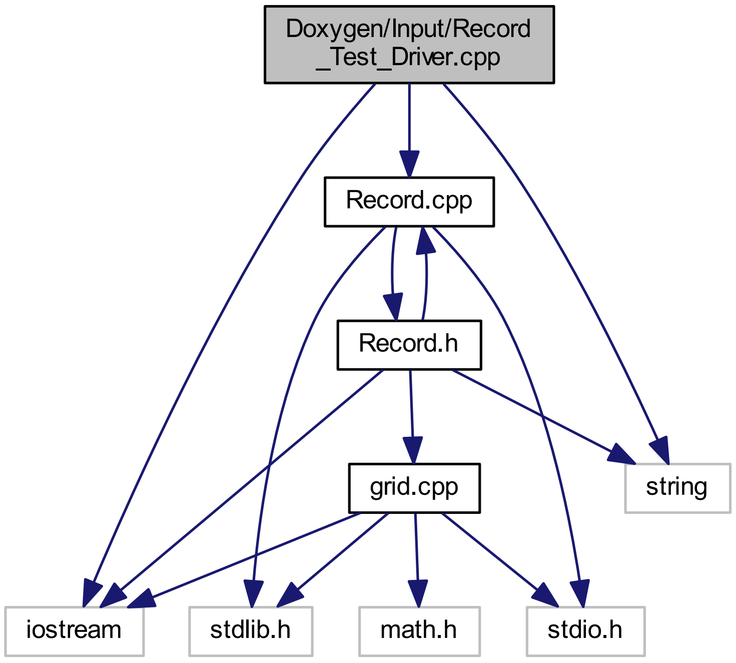
```

#ifndef RECORD_H
#define RECORD_H

#include <iostream>
#include <string>
#include "grid.cpp"
using namespace std;

class Record
{
  public:
    Record();
    Record(string, string, string, string, Grid);
    Record(string, string, string, string, string, string);
    void display();
    void display(string); //This might benefit from calling get_field
    string get_field(string); //This should have a switch statement
    void set_field(string, string);
    void set_longitude_latitude(float, float);
    void set_grid_point(Grid);

  private:
    bool isEmpty;
    string zip_code;
    string place_name;
    string state;
    string county;
    Grid gridPoint;
    float string_to_float(string);
};

#include "Record.cpp"

#endif
```

4.15 Doxygen/Input/Record_Test_Driver.cpp File Reference

```
#include "Record.cpp"
#include <iostream>
```

```
#include <string>
```

Include dependency graph for Record_Test_Driver.cpp:



Enumerations

- enum Field {
  Z, ZIP, CITY, P,
  PLACE_NAME, STATE, COUNTY, G,
  GRID }

Functions

- int main ()

4.15.1 Enumeration Type Documentation

4.15.1.1 Field

```
enum Field
```

Enumerator

Z	
ZIP	
CITY	
P	
PLACE_NAME	
STATE	
COUNTY	
G	
GRID	

Definition at line 9 of file Record_Test_Driver.cpp.

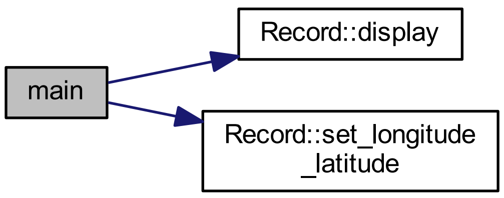
4.15.2 Function Documentation

4.15.2.1 main()

```
int main ( )
```

Definition at line 22 of file Record_Test_Driver.cpp.

Here is the call graph for this function:



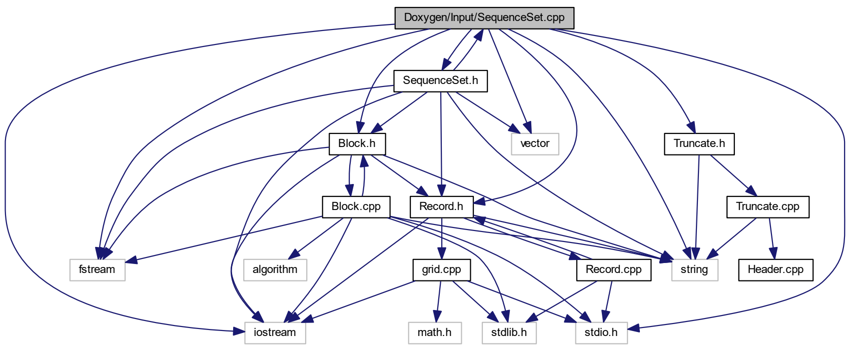
4.16 Record_Test_Driver.cpp

```
//g++ -std=c++11  -o record_test Record_Test_Driver

#include "Record.cpp"
#include<iostream>
#include<string>

using namespace std;

enum Field
{
    Z,
    ZIP,
    CITY,
    P,
    PLACE_NAME,
    STATE,
    COUNTY,
    G,
    GRID
};

int main()
{
  //test default constructor
    Record testRecord;
  cout « "Default constructor record:";
  testRecord.display();

  //test fill record
  string zip = "56345";
  string place = "Little Falls";
  string state = "Minnesota";
  string county = "Morrison";
  float longitude = 74.25;
  float latitude = 79.72;

  testRecord.set_zip_code( zip );
  testRecord.set_place_name( place );
  testRecord.set_state( state );
  testRecord.set_county( county );
  testRecord.set_longitude_latitude( longitude, latitude );

  cout « "Filled Record:";
  testRecord.display();

  //test constructor 2
  string longitude_string = "74.25";
  string latitude_string = "79.72";

  Record testRecord2(zip, place, state, county, longitude_string, latitude_string);

  cout « "Constructor2 record (long/lat are strings):";
  testRecord2.display();

    //test constructor 3
  Grid grid_test(longitude, latitude);

  Record testRecord3(zip, place, state, county, grid_test);

  cout « "Constructor3 record (long/lat are gridPoint):";
  testRecord3.display();

  cout « endl « "check enum:";
  testRecord3.display(CITY);
  cout « " expected: Little Falls" « endl;

  testRecord3.display(STATE);
  cout « " expected: Minnesota" « endl;

    return 0;
}
```

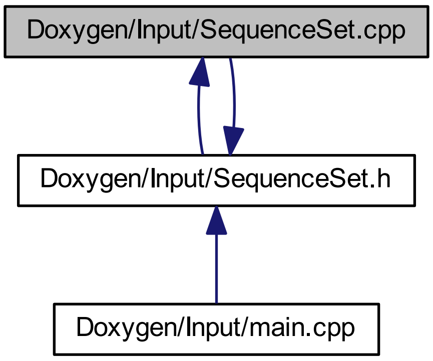
4.17 Doxygen/Input/SequenceSet.cpp File Reference

```
#include "SequenceSet.h"
#include <iostream>
#include "Truncate.h"
#include "Record.h"
#include "Block.h"
#include <string>
#include <fstream>
#include <vector>
#include <stdio.h>
```

Include dependency graph for SequenceSet.cpp:



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



Functions

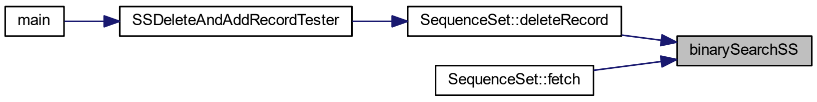
- int binarySearchSS (const string arr[ ], string x, int n)

4.17.1 Function Documentation

4.17.1.1 binarySearchSS()

```
int binarySearchSS (
             const string arr[],
             string x,
             int n )
```

Here is the caller graph for this function:



4.18 SequenceSet.cpp

```

#include "SequenceSet.h"
#include <iostream>
#include "Truncate.h"
#include "Record.h"
#include "Block.h"
#include "SequenceSet.h"
#include <string>
#include <fstream>
#include <vector>
#include <stdio.h>

using namespace std;

//binarySearch recycled from block
int binarySearchSS(const string arr[], string x, int n);

SequenceSet::SequenceSet(){
  ofstream SSFile;
  SSFile.open(SSFileName);
  SSFile « "Sequence Set File\n";
  SSFile.close();
  recordCount = getRecordCount();
  fillIndex();
  Block * currentBlock = headBlock;
  blockCount = 0;
  for(unsigned long long i = 0; i < recordCount; i++){
    if(i%BLOCKFILLCOUNT == 0 && i != 0){
        if(DEBUG){cout « "Making a new block for the chain." « endl;}
      blockCount++;
      Block * newBlock = new Block(blockCount);
      currentBlock->setNextBlock(newBlock);
      newBlock->setPrevBlock(currentBlock);
      currentBlock = newBlock;
    }
    if(DEBUG){cout«"Passing "«to_string(pKeyIndex.at(i))«" into the add function."«endl;}
    currentBlock->addRecord(to_string(pKeyIndex.at(i)));
  }
  writeBlocks();

  //reset the record avail list
  ofstream recordAvailList;
  recordAvailList.open(recordAvailListFileName);
  recordAvailList « "";
  recordAvailList.close();
}

unsigned long long SequenceSet::headerLength(string _fileName){
  fstream data;
  unsigned long long length = 0;
  unsigned long long L = 0;
  data.open(_fileName);
  string str;

  if(DEBUG){cout « "String outside while loop, in headerLength: " « str « endl;}
  while(data.peek() != EOF){
    if(DEBUG && false){cout « "String in headerLength: " « str « endl;}
    getline(data, str);
    length += str.length();
    length++;
    if(str == HEADERENDSTRING){
      L = length;
      if(DEBUG){cout«"L defined: "« L «"\n";}
    }
  }

  data.close();

  return L;
}

unsigned int SequenceSet::getRecordCount(){
    string fileName = DATAFILENAME;
    string field;
    string str = "";
    char c;
    fstream data;
    unsigned int recordCount = 0;
    data.open(fileName);
    getline(data, field); //Skip title
    while(data.peek() != ':' ){
        data.get(c);
        field += c;
        if(DEBUG && true){cout « "Char c: " « c « endl;}
    }

    //This while is to skip non number values before approaching what to do with the values
    while(data.peek() < '0' || data.peek() > '9' ){
        data.get(c);
    }
    getline(data, str);

    recordCount = stoi(str);

    if(DEBUG) {cout « "String: " « str « "\nrecords: " « recordCount « endl;}
    if(field == "Records"){
        getline(data, str);
        recordCount = stoi(str);
        if(DEBUG){cout « "Record Count: " « recordCount « endl;}
    }
    data.close();

    return recordCount;
}

void SequenceSet::fillIndex(){
    string field;
    string str = "";
    char c;
    fstream data;

    data.open("RecordOffsets.txt");

    for(unsigned int i = 0; i < recordCount; i++){
        string recordData = "";
        getline(data, recordData);
        if(DEBUG){cout « "recordData: " « recordData « endl;}
        str = "";
        for(int j = 0; j < ZIPLENGTH; j++){
          str += recordData[j];
        }
        //index[i][0] = stoi(str);  //five chars of string
        pKeyIndex.push_back(stoi(str));
        if(DEBUG){cout « "String: " « str « endl;}
        if(DEBUG){cout « "pKeyIndex.at(i): " « pKeyIndex.at(i) «endl;}
        str = "";
        for(int j = ZIPLENGTH; j < recordData.length(); j++){
          str += recordData[j];
        }
        //index[i][1] = stoi(str);  //the rest of the string
        offsetIndex.push_back(stoi(str));
        if(DEBUG){cout « "String: " « str « endl;}
        if(DEBUG){cout « "offsetIndex.at("«i«"): " « offsetIndex.at(i) «endl;}
    }
    data.close();
}

string SequenceSet::fetch(string pKey){
  fstream data;
  data.open(DATAFILENAME);
  string returnString = "";
  for(int i = ZIPLENGTH - pKey.length(); i > 0; i--){
    if(DEBUG){cout « "For loop in fetch. i = " « i « endl;}
    returnString += " ";
  }
  returnString = pKey;
  returnString += " not found.\n";

  int position;
  if(pKey != ""){
    position = binarySearchSS(pKey);
  }
  if(DEBUG) {cout « "Searching "« pKey « " returned: " « position « endl;}
  if(position>=0 && pKey != ""){
    data.seekg(offsetIndex.at(binarySearchSS(pKey)));
    getline(data, returnString);
  }
  data.close();

  return returnString;
}

string SequenceSet::fetch(unsigned int pKey){
  return fetch(to_string(pKey));
}

void SequenceSet::makeRecordOffsets(string fileName){
    string zip = "      ";
    fstream data;
    ofstream index;
    string str;
    index.open("RecordOffsets.txt");
    unsigned long long offset = headerLength(fileName);
    data.open(fileName);
    data.seekg(offset);

    if(DEBUG && false){cout « "String in makeRecordOffsets is: " « str « endl;}
    getline(data, str);

    while(data.peek()!=EOF){
      if (DEBUG && false){cout « str « endl;}
    for(int i = 0; i < ZIPLENGTH; i++){
      zip[i] = str[i];
    }
    if(DEBUG && false){cout«zip« " is at " « offset «endl;}
    index « zip « offset « endl;
    getline(data, str);
    offset += str.length();
    offset++;
    }

    data.close();
    index.close();
}

int SequenceSet::binarySearchSS(string x)
{
    //int int_arr[n];
    unsigned int n = recordCount;
    int int_string;
/*
    //convert the records (array of strings) to array of int
    for (unsigned int i = 0; i < n; i++)
    {
    if(arr[i] != null_str)
          int_arr[i] = stoi(arr[i]);
    }
*/
    //convert string to find to int
    if(DEBUG){cout « "(stoi)ing this string: \"" « x « "\"\n";}
    try{
    int_string = stoi(x);    unsigned int l = 0 ;
    unsigned int r = n - 1;
    while (l <= r)
    {
      int m = l + (r - l) / 2;
          if(DEBUG) {cout « "mid: " « m «endl;}

      //if(DEBUG) {cout « "comparing " « int_string « " and " « int_arr[m] «endl;}
      if(DEBUG) {cout « "comparing " « int_string « " and " « pKeyIndex.at(m) «endl;}

        if ( pKeyIndex.at(m) == int_string ){
        if(DEBUG) {cout « "record found" «endl;}
        return m;
          }

      // If x is greater, ignore left half
      if ( pKeyIndex.at(m) < int_string ){
        l = m + 1;
        if(DEBUG) {cout « "new l: " « l «endl;}
      }

        // If x is smaller, ignore right half
      else{
        r = m - 1;
        if(DEBUG) {cout « "new r: " « l «endl;}
```

```cpp
      }
    }
}
    catch(...){cout « "ERROR (stoi)ING THIS STRING: \"" « x « "\"\n";}

    return -1;
}

Record SequenceSet::fillRecord(string RecordString){
  string zip_code, place_name, state, county, latitude, longitude;
  int position = 0;
  if(DEBUG){cout  « "In fillRecord for Sequence Set Class\n\tRecordString: "
                  « RecordString « endl;}
  zip_code = "";
  for(auto i = 0; i < ZIPLENGTH    ; i++){
    if(RecordString[position] != ' '){
      zip_code += RecordString[position];
    }
    position++;
  }

  place_name = "";
  for(int i = 0; i < 31/*Length of place name*/; i++){
    if(RecordString[position] != ' '){
      place_name += RecordString[position];
    }
    position++;
  }

  state = "";
  for(int i = 0; i < 2/*Length of state*/; i++){
    if(RecordString[position] != ' '){
      state += RecordString[position];
    }
    position++;
  }

  county = "";
  for(int i = 0; i < 38/*Length of county*/; i++){
    if(RecordString[position] != ' '){
      county += RecordString[position];
    }
    position++;
  }

  latitude = "";
  for(int i = 0; i < 9/*Length of latitude*/; i++){
    if(RecordString[position] != ' '){
      latitude += RecordString[position];
    }
    position++;
  }

  longitude = "";
  for(int i = 0; i < 8/*Length of longitude*/; i++){
    if(RecordString[position] != ' '){
      longitude += RecordString[position];
    }
    position++;
  }
  if(DEBUG){cout  « "\tRecordElements: " « "\n\t\t"
                  « zip_code « "\n\t\t" « place_name « "\n\t\t"
                      « state « "\n\t\t" « county « "\n\t\t"
                  « latitude « "\n\t\t" « longitude « endl;}

  Record returnRecord(zip_code, place_name, state, county, latitude, longitude);

  if(DEBUG){returnRecord.display();}

  return returnRecord;
}

void SequenceSet::writeBlocks(){
  Block * currentBlock = headBlock;
  for(auto i = 0; i < blockCount; i ++){
    if(DEBUG){cout « "Writing block "« i «" from the chain." « endl;}
    currentBlock->write(SSFileName);
    currentBlock = currentBlock->getNextBlock();
  }
}

void SequenceSet::fillRecordBlock(unsigned long long blockID){
  string str, zip, passed;
  Block * currentBlock = headBlock;
  for(auto i = 0; i < blockID; i++){
    currentBlock = currentBlock->getNextBlock();
  }

  currentBlock->getRecords(recordBlock);
  for(auto i = 0; i < currentBlock->getRecordCount(); i++){
    passed = fetch(recordBlock[i].get_field("ZIP"));
    if(DEBUG){
      cout  « "\n*******************************************"
            « "\nString passed to fill record: " « passed « endl;
    }
    if(passed != " not found.\n" && passed != " not found."){
      recordBlock[i] = fillRecord(passed);
      if(DEBUG){recordBlock[i].display();}
    }
  }
}

void SequenceSet::addBlockStateKey(unsigned long long blockID){
  fillRecordBlock(blockID);
  Block * currentBlock = headBlock;

  for(auto i = 0; i < blockID; i++){
    currentBlock = currentBlock->getNextBlock();
  }

  for(auto i = 0; i < currentBlock->getRecordCount(); i++){
    string state = recordBlock[i].get_field("state");

    for(auto i = 0; i < RECORDSPERBLOCK; i++){
      string state = recordBlock[i].get_field("state");

      if(state != ""){
    bool stateFound = false;
    unsigned int index = 0;

    if(stateZips.size() == 0){
            vector <string> newRow;
            newRow.push_back(state);
            stateZips.push_back(newRow);
    }

    while(index < stateZips.size() && !stateFound){
        if(stateZips[index].at(0) == state){
            if(DEBUG){cout « "Found " « state « " at index = " « index « endl;}
            stateFound = true;
        }
        else{index++;}
    }

    if(!stateFound){
        if(DEBUG){cout « state«" not found.\n";}
            vector <string> newRow;
            newRow.push_back(state);
            stateZips.push_back(newRow);
        if(DEBUG){cout « stateZips[index].at(0)«" pushed successfully.\n";}
        if(DEBUG){
            stateZips[index].push_back(":)");
            cout « "Pushing a smily :)\n";
            cout « stateZips[index].at(1) « endl;
            stateZips[index].pop_back();
        }
    }

    if(DEBUG){cout « "Pushing " « recordBlock[i].get_field("zip") «" to "« index «" column.\n";}
    stateZips[index].push_back(recordBlock[i].get_field("zip"));
    //if(DEBUG){cout « stateZips[index].at(stateZips[index].size())«" pushed successfully.\n";}


    cout    « stateZips[index].at(0) « ": "
        « stateZips[index].at(stateZips[index].size()-1) « endl;
      }
    }
  }
}


bool SequenceSet::deleteRecord(int pKey)
{
  //search if the record is in the sequence set
  int position = binarySearchSS( to_string(pKey) );
  if(DEBUG) {cout « "Searching for "« pKey « " returned: " « position « endl;}
  if(position == -1){
    cout « "Record does not exist in Sequence Set." « endl;
    return false;
  }
  else{
    //add deleted record offset to avail list
    string strTemp = "";
    string newString = "";
```

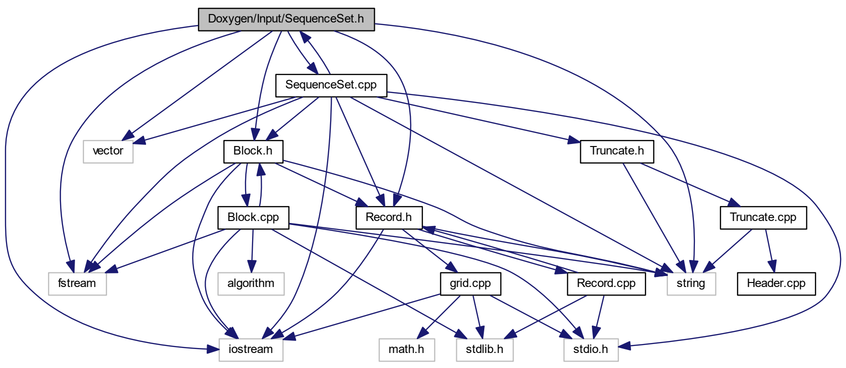
```cpp
    fstream recordAvailListIn;
    recordAvailListIn.open(recordAvailListFileName);
    while(recordAvailListIn.peek() != EOF){
        strTemp += recordAvailListIn.get();
        cout « strTemp « endl;
    }
    newString = to_string( offsetIndex.at(position) ) + "/" + to_string( position ) + "\n" + strTemp;
    cout « newString « " result" « endl;
    recordAvailListIn.close();

    ofstream recordAvailList;
    recordAvailList.open(recordAvailListFileName);
    recordAvailList « newString;
    recordAvailList.close();

    //delete record from us_postal_codes.txt
    fstream usPostalCodes;
    usPostalCodes.open("us_postal_codes.txt");
    usPostalCodes.seekg(offsetIndex.at(position));
    for(int i = 0; i < 94; i++){ //94 is the length of record
      usPostalCodes « " ";
    }
    usPostalCodes.close();

    //delete record in index vector
    pKeyIndex.erase(pKeyIndex.begin() + position);
    offsetIndex.erase(offsetIndex.begin() + position);
      if(DEBUG) {position = binarySearchSS( to_string(pKey) );}
    if(DEBUG) {cout « "Deleted record in index vector. Researching for "« pKey « " returned: " « position « endl;}
    recordCount--; //decrement the total record count

    //delete record in linked list of blocks
    Block * currentBlock = headBlock;
    for(auto i = 0; i < blockCount; i ++){
      if(DEBUG){cout « "Searching block "« i «" from the chain." « endl;}
      if( pKey <= currentBlock->getLastRecordPKey() ){
        currentBlock->deleteRecord( to_string(pKey) );
        break;
      }
      else{
        currentBlock = currentBlock->getNextBlock();
      }
    }

    //merge blocks if needed
    if( currentBlock->getRecordCount() < RECORDSPERBLOCK / 2 ){
      //check next block to see if it can merge
      if( (currentBlock->getNextBlock())->getRecordCount() == RECORDSPERBLOCK / 2 ){
        currentBlock->getRecords( recordBlock ); //get the pkeys
        for(int i = 0; i < currentBlock->getRecordCount(); i++){
          (currentBlock->getNextBlock())->addRecord(recordBlock[i].get_field("zip"));
          currentBlock->deleteRecord( recordBlock[i].get_field("zip") );
        }
          //add the pointer to the current block to the avail vector
          blockAvailList.push_back( currentBlock );
          //change the pointers to avoid the empty block
          currentBlock->getPreviousBlock()->setNextBlock( currentBlock->getNextBlock() );
          currentBlock->getNextBlock()->setPrevBlock( currentBlock->getPreviousBlock() );
          blockCount--;
      }
      //check if previous block can merge
      else if( (currentBlock->getPreviousBlock())->getRecordCount() == RECORDSPERBLOCK / 2 ){
        currentBlock->getRecords( recordBlock ); //get the pkeys
        for(int i = 0; i < currentBlock->getRecordCount(); i++){
          (currentBlock->getPreviousBlock())->addRecord(recordBlock[i].get_field("zip"));
          currentBlock->deleteRecord( recordBlock[i].get_field("zip") );
        }
          //add the pointer to the current block to the avail vector
          blockAvailList.push_back( currentBlock );
          //change the pointers to avoid the empty block
          currentBlock->getPreviousBlock()->setNextBlock( currentBlock->getNextBlock() );
          currentBlock->getNextBlock()->setPrevBlock( currentBlock->getPreviousBlock() );
          blockCount--;
      }
      //check if next block can redistribute
      else if( (currentBlock->getNextBlock())->getRecordCount() > RECORDSPERBLOCK / 2 ){
        (currentBlock->getNextBlock())->getRecords( recordBlock ); //get the pkeys
        currentBlock->addRecord( recordBlock[0].get_field("zip") );
        (currentBlock->getNextBlock())->deleteRecord( recordBlock[0].get_field("zip") );
      }
      //check if previous block can redistribute ???Will Never Happen??????????????????????????????????????????????????????????????????????????????????????
      // else if( (currentBlock->getPreviousBlock())->getRecordCount() > RECORDSPERBLOCK / 2 ){
      //   (currentBlock->getPreviousBlock())->getRecords( recordBlock ); //get the pkeys
      //   currentBlock->addRecord( recordBlock[0].get_field("zip") );
      //   (currentBlock->getNextBlock())->deleteRecord( recordBlock[0].get_field("zip") );
      //}
    }

    return true;
  }
}

string SequenceSet::extremeCoord(string state, char direction)
{
  direction = toupper(direction);
    float extremePoint = 0;
    string zip = "";
  Record currentRecord;
  string str = state;

  bool found = false;
  unsigned int index = 0;
  while(index < stateZips.size() && !found){
    if(stateZips[index][0] == str){found = true;}
    else{index++;}
  }
  currentRecord = fillRecord(fetch(stateZips[index][1]));

    switch(direction)
    {
        case 'N':
        {
      extremePoint = stof(currentRecord.get_field("Lat"));
      zip = currentRecord.get_field("zip");
            for(int i = 1; i < stateZips[index].size(); i++)
            {
                currentRecord = fillRecord(fetch(stateZips[index][i]));
                if(extremePoint < stof(currentRecord.get_field("Lat")))
                {
          zip = currentRecord.get_field("zip");
                    extremePoint = stof(currentRecord.get_field("Lat"));
                }
            }
        }
        break;

        case 'E':
        {
            extremePoint = stof(currentRecord.get_field("Long"));
            zip = currentRecord.get_field("zip");
            for(int i = 1; i < stateZips[index].size(); i++)
            {
                currentRecord = fillRecord(fetch(stateZips[index][i]));
                if(extremePoint < stof(currentRecord.get_field("Long")))
                {
          zip = currentRecord.get_field("zip");
                    extremePoint = stof(currentRecord.get_field("Long"));
                }
            }
        }
        break;

        case 'S':
        {
            extremePoint = stof(currentRecord.get_field("Lat"));
            zip = currentRecord.get_field("zip");
            for(int i = 1; i < stateZips[index].size(); i++)
            {
                currentRecord = fillRecord(fetch(stateZips[index][i]));
                if(extremePoint > stof(currentRecord.get_field("Lat")))
                {
          zip = currentRecord.get_field("zip");
                    extremePoint = stof(currentRecord.get_field("Lat"));
                }
            }
        }
        break;

        case 'W':
        {
            extremePoint = stof(currentRecord.get_field("Long"));
            zip = currentRecord.get_field("zip");
            for(int i = 1; i < stateZips[index].size(); i++)
            {
                currentRecord = fillRecord(fetch(stateZips[index][i]));
                if(extremePoint > stof(currentRecord.get_field("Long")))
                {
          zip = currentRecord.get_field("zip");
                    extremePoint = stof(currentRecord.get_field("Long"));
                }
            }
        }
        break;

    default:
    {
      cout « "UNDEFINED OPTION\n";
    }
    }
    return zip;
}

int SequenceSet::test(){
    string field;
    string str = "";
    char c;
    fstream data;

    int randomRecord = rand() % recordCount;
    //cout « "Retrieving record: " « index[randomRecord][0] « endl;
    cout « "Retrieving record: " « pKeyIndex.at(randomRecord) « endl;
    data.open(DATAFILENAME);
    //data.seekg(index[randomRecord][1]);
    data.seekg(offsetIndex.at(randomRecord));
    getline(data, str);
    cout « str « endl;

    cout « fetch(1721) « endl;
    fillRecordBlock(88);

    for(auto i = 0; i < RECORDSPERBLOCK; i++){
      if(DEBUG){cout «"\n***********************************\n";}
      recordBlock[i].display();
    }

    sKeyStateBuilder();

    unsigned int index = 0;
    unsigned int record = 1;
    Record currentRecord;

    str = "MN";
    bool found = false;
    while(index < stateZips.size() && !found){
      if(stateZips[index][0] == str){found = true;}
      else{index++;}
    }

    while(record < stateZips[index].size()){
      str = fetch(stateZips[index][record]);
      cout « str « endl;
      currentRecord = fillRecord(str);
      currentRecord.display();
      record++;
    }

    cout « extremeCoord(str, 'n') « endl;

    return 0;
}


void SequenceSet::sKeyStateBuilder(){
    if(DEBUG){cout « "Building sKeys for states.\n";}
    Block * currentBlock = headBlock;
    unsigned int index = 0;
    while(currentBlock!=NULL){
      addBlockStateKey(index);
      currentBlock = currentBlock->getNextBlock();
      index++;
    }
}

void SequenceSet::addRecord(Record record)
{
    //search record in linked list of blocks
    Block * currentBlock = headBlock;
    for(auto i = 0; i < blockCount; i ++){
        if(DEBUG){cout « "Searching block "« i «" from the chain." « endl;}
        if( stoi( record.get_field("zip") ) <= currentBlock->getLastRecordPKey() ){ //find the right block
      if(currentBlock->getRecordCount() == RECORDSPERBLOCK){ //if the block is full, do block splitting
        if( !blockAvailList.empty() ){  //if there exists a current empty block
          Block* tempBlockPtr = blockAvailList.back(); //get the pointer to the empty block
          blockAvailList.pop_back();  //delete the pointer from the avail list
          //add the relative block to the linked list
          tempBlockPtr->setNextBlock( currentBlock->getNextBlock() );
          tempBlockPtr->setPrevBlock( (currentBlock->getNextBlock())->getPreviousBlock() );
          (currentBlock->getNextBlock())->setPrevBlock(tempBlockPtr);
          currentBlock->setNextBlock(tempBlockPtr);
          //split the data into the new block number
          currentBlock->getRecords( recordBlock ); //get the pkeys
          for(int i = RECORDSPERBLOCK / 2; i < RECORDSPERBLOCK; i++){
            (currentBlock->getNextBlock())->addRecord(recordBlock[i].get_field("zip"));
            currentBlock->deleteRecord( recordBlock[i].get_field("zip") );
          }
          //add the new record to the block
          currentBlock->addRecord( record.get_field("zip") );
          blockCount++;
          break; //stop searching through linked list of blocks
        }
        else{  //if a current empty block doesn’t exist, create a new block......
          Block* newBlockPtr = new Block;
          newBlockPtr->setRBN(blockCount);
          newBlockPtr->setNextBlock( currentBlock->getNextBlock() );
          newBlockPtr->setPrevBlock( (currentBlock->getNextBlock())->getPreviousBlock() );
          (currentBlock->getNextBlock())->setPrevBlock(newBlockPtr);
          currentBlock->setNextBlock(newBlockPtr);
          //split the data into the new block number
          currentBlock->getRecords( recordBlock ); //get the pkeys
          for(int i = RECORDSPERBLOCK / 2; i < RECORDSPERBLOCK; i++){
            (currentBlock->getNextBlock())->addRecord(recordBlock[i].get_field("zip"));
            currentBlock->deleteRecord( recordBlock[i].get_field("zip") );
          }
          //add the new record to the block
          currentBlock->addRecord( record.get_field("zip") );
          blockCount++;
          break; //stop searching through linked list of blocks
        }
      }
      else{
        currentBlock->addRecord( record.get_field("zip") );
      }
            break; //stop searching through linked list of blocks
        }
        else{
            currentBlock = currentBlock->getNextBlock();
        }
    }

  //add record to us_postal_codes.txt
  fstream recordAvailList;
  string str = "";
  string strTemp = "";
  string offset = "";
  string position = "";
  recordAvailList.open(recordAvailListFileName);
  if( recordAvailList.peek() != EOF ){ //if recordAvailList is not empty
    fstream usPostalCodes;
    usPostalCodes.open("us_postal_codes.txt");
    getline(recordAvailList, str); //get the offset and vector position from avail list
    int i = 0;
    while( str[i] != '/' ){ //parse the offset from the string
        offset += str[i];
        cout « offset « endl;
        i++;
    }
    i++;
    while( i < str.length()  ){ //parse the position from the string
        position += str[i];
        cout « position « endl;
        i++;
    }
    writeToTxt(record, offset, "us_postal_codes.txt");
    usPostalCodes.close();
    recordAvailList.close();
    //update the recordAvailList
    recordAvailList.open(recordAvailListFileName);
    str += "\n";
    cout « str « " str to delete" « endl;
    while(recordAvailList.peek() != EOF){
        strTemp += recordAvailList.get();
        cout « strTemp « endl;
        if(strTemp == str){
            strTemp = "";
        }
    }
    recordAvailList.close();
    remove("availRecordList.txt");
    ofstream recordAvailListOut;
    recordAvailListOut.open(recordAvailListFileName, ios::app);
    cout « strTemp « " result" « endl;
    recordAvailListOut « strTemp;
    recordAvailListOut.close();
    //add record to index vector
```

```
    for(int i=0; i<20; ++i)
        std::cout « pKeyIndex[i] « ' ';
    pKeyIndex.insert(pKeyIndex.begin() + stoi( position ), stoi( record.get_field("zip") ) );
    offsetIndex.insert(offsetIndex.begin() + stoi( position ), stoi( offset ) );
    cout «endl;
    for(int i=0; i<20; ++i)
        std::cout « pKeyIndex[i] « ' ';
  }
  else{ //if recordAvailList is empty
    unsigned int nextOffset = offsetIndex.back() + 95;//95 is record length+1
    cout « nextOffset « " nextoffset" « endl;
    pKeyIndex.push_back( stoi( record.get_field("zip") ) );
    offsetIndex.push_back( nextOffset );
    writeToTxt(record, to_string( nextOffset ), "us_postal_codes.txt");
    ofstream usPostalCodes;
    usPostalCodes.open("us_postal_codes.txt", ios::app);
    usPostalCodes « endl;
    usPostalCodes.close();
  }
}

void SequenceSet::rewriteSSFile()
{
    //rewrite the squence set file with missing record
    remove("Sequence_Set.txt");
    ofstream SSFile;
    SSFile.open(SSFileName);
    SSFile « "Sequence Set File\n";
    SSFile.close();
    writeBlocks();
}

//write the record to the postal codes file
void SequenceSet::writeToTxt(Record record, string offset, string _fileName)
{
    fstream data;
    data.open(_fileName);
    data.seekg( stoi( offset ) );

    string dataString = "";
    string totalString = "";

    dataString = record.get_field("Zip");
    int fieldLength = 6;
    for(int i = 0; i < fieldLength - dataString.length(); i++){
        totalString += " ";
    }
    totalString += dataString;

    dataString = record.get_field("city");
    fieldLength = 31;
    totalString += dataString;
    for(int i = 0; i < fieldLength - dataString.length(); i++){
        totalString += " ";
    }

    dataString = record.get_field("state");
    totalString += dataString;

    dataString = record.get_field("county");
    fieldLength = 38;
    totalString += dataString;
    for(int i = 0; i < fieldLength - dataString.length(); i++){
        totalString += " ";
    }

    dataString = record.get_field("long");
    while(dataString.length() > 8){
        dataString.pop_back();
    }
    fieldLength = 8;
    for(int i = 0; i < fieldLength - dataString.length(); i++){
        totalString += " ";
    }
    totalString += dataString;

    dataString = record.get_field("lat");
    fieldLength = 9;
    while(dataString.length() > 9){
        dataString.pop_back();
    }
    for(int i = 0; i < fieldLength - dataString.length(); i++){
        totalString += " ";
    }
    totalString += dataString;

    data « totalString;

    data.close();
}
```

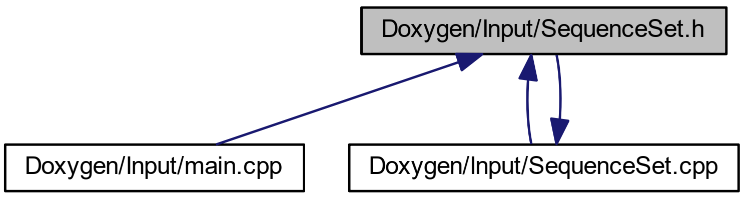
4.19 Doxygen/Input/SequenceSet.h File Reference

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include "Block.h"
#include "Record.h"
#include "SequenceSet.cpp"
```

Include dependency graph for SequenceSet.h:



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



Classes

- class SequenceSet

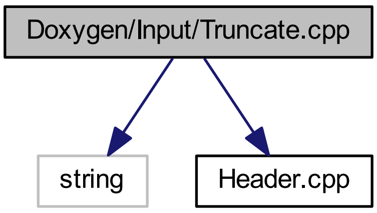
4.20 SequenceSet.h

```

#ifndef SEQUENCESET_H
#define SEQUENCESET_H

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

#include "Block.h"
#include "Record.h"

using namespace std;

class SequenceSet
{
private:
    string SSFileName = "Sequence_Set.txt";
    string recordAvailListFileName = "availRecordList.txt";
    unsigned long long headerLength(string);
    unsigned long long blockCount;
    unsigned int recordCount;
    //unsigned int indexArray[getRecordCount()][2];
    vector<unsigned int>pKeyIndex;
    vector<unsigned int>offsetIndex;
    vector<vector<string»stateZips;
    vector<vector<string»sKeyCounty;
    vector<vector<string»sKeyPlace;
    vector<Block*>blockAvailList;
    Record recordBlock[RECORDSPERBLOCK];
    Block * headBlock = new Block;

    int binarySearchSS(string x);

public:
    SequenceSet();

    void makeRecordOffsets(string fileName);

    void fillIndex();

    void fillRecordBlock(unsigned long long blockID);

    void writeBlocks();

    Record fillRecord(string RecordString);

    unsigned int getRecordCount();

    string fetch(string pKey);

    string fetch(unsigned int pKey);

    void addBlockStateKey(unsigned long long blockID);

    void sKeyStateBuilder();

        string extremeCoord(string, char);

    int test();

    bool deleteRecord(int pKey);
    void addRecord(Record record);
    void rewriteSSFile();
    void writeToTxt(Record, string, string);
};

#include "SequenceSet.cpp"

#endif

```

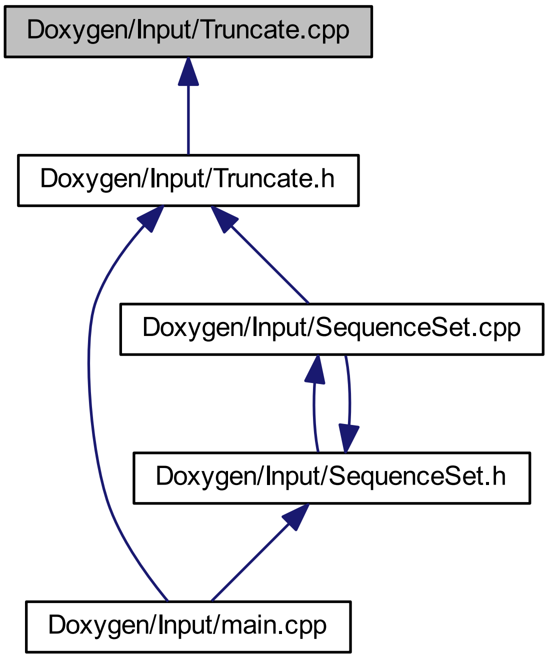
4.21 Doxygen/Input/Truncate.cpp File Reference

```
#include <string>
#include "Header.cpp"
```

Include dependency graph for Truncate.cpp:



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


Doxygen/Input/Truncate.cpp
Doxygen/Input/Truncate.h
Doxygen/Input/SequenceSet.cpp
Doxygen/Input/SequenceSet.h
Doxygen/Input/main.cpp

4.21.1 Detailed Description

Author

Christenson, Mark

Definition in file Truncate.cpp.

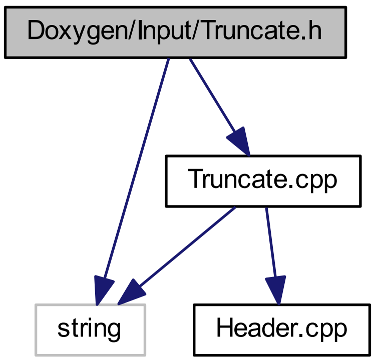
4.22 Truncate.cpp

```
//#include "Truncate.h"
#include <string>
#include "Header.cpp"

Truncate::Truncate(){
    if (true)
    {
        cout « "Truncate object made";
    }
}
Truncate::Truncate(int _size){
    maxLength = _size;
}

string Truncate::truncatedString(string _string) {
    string newStr = _string;
    newStr.resize(maxLength);

    return newStr;
}

string Truncate::modifyString(string & _originalStr) {
    _originalStr.resize(maxLength);

    return _originalStr;
}
```

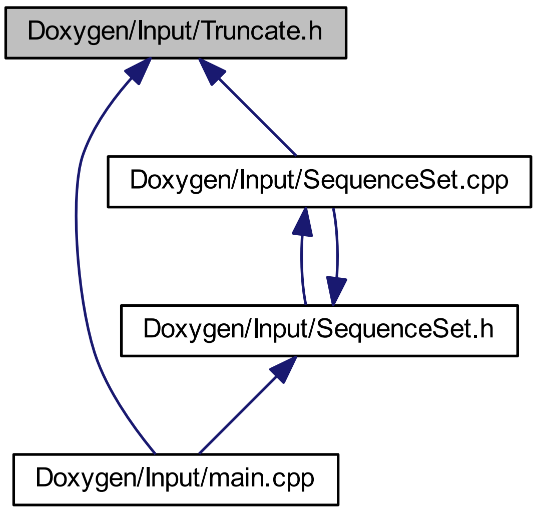
4.23 Doxygen/Input/Truncate.h File Reference

```
#include <string>
#include "Truncate.cpp"
```

Include dependency graph for Truncate.h:



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



Classes

- class Truncate

4.24 Truncate.h

```
#ifndef TRUNCATE_H
#define TRUNCATE_H
#include <string>

using namespace std;

class Truncate {
public:
    Truncate();
    Truncate(int);
    string modifyString(string&);
    string truncatedString(string);
private:
    int maxLength = 10;
};

#include "Truncate.cpp"

#endif
```

Index