

Sequence Set

1.0

Generated by Doxygen 1.8.16

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 Block Class Reference	5
3.1.1 Detailed Description	6
3.1.2 Constructor & Destructor Documentation	6
3.1.2.1 Block() [1/4]	6
3.1.2.2 Block() [2/4]	6
3.1.2.3 Block() [3/4]	7
3.1.2.4 Block() [4/4]	7
3.1.3 Member Function Documentation	7
3.1.3.1 addRecord()	8
3.1.3.2 blockData()	8
3.1.3.3 deleteRecord()	9
3.1.3.4 getLastRecordPKey()	9
3.1.3.5 getNextBlock()	10
3.1.3.6 getPreviousBlock()	10
3.1.3.7 getRBN()	10
3.1.3.8 getRecordCount()	11
3.1.3.9 getRecords()	11
3.1.3.10 search()	12
3.1.3.11 setNextBlock()	13
3.1.3.12 setPrevBlock()	13
3.1.3.13 setRBN()	14
3.1.3.14 write()	14
3.2 Grid Class Reference	15
3.2.1 Detailed Description	15
3.2.2 Constructor & Destructor Documentation	15
3.2.2.1 Grid() [1/2]	15
3.2.2.2 Grid() [2/2]	16
3.2.3 Member Function Documentation	16
3.2.3.1 getDistance()	16
3.2.3.2 getLatitude()	17
3.2.3.3 getLongitude()	17
3.2.3.4 setLatitude() [1/2]	18
3.2.3.5 setLatitude() [2/2]	18
3.2.3.6 setLongitude() [1/2]	19
3.2.3.7 setLongitude() [2/2]	19
3.3 Record Class Reference	19

3.3.1 Detailed Description	20
3.3.2 Constructor & Destructor Documentation	20
3.3.2.1 Record() [1/3]	20
3.3.2.2 Record() [2/3]	21
3.3.2.3 Record() [3/3]	21
3.3.3 Member Function Documentation	21
3.3.3.1 display() [1/2]	22
3.3.3.2 display() [2/2]	22
3.3.3.3 get_field()	23
3.3.3.4 set_field()	23
3.3.3.5 set_grid_point()	24
3.3.3.6 set_longitude_latitude()	24
3.4 SequenceSet Class Reference	25
3.4.1 Detailed Description	26
3.4.2 Constructor & Destructor Documentation	26
3.4.2.1 SequenceSet()	26
3.4.3 Member Function Documentation	26
3.4.3.1 addBlockStateKey()	27
3.4.3.2 addRecord()	27
3.4.3.3 deleteRecord()	28
3.4.3.4 extremeCoord()	29
3.4.3.5 fetch() [1/2]	30
3.4.3.6 fetch() [2/2]	31
3.4.3.7 fillIndex()	31
3.4.3.8 fillRecord()	31
3.4.3.9 fillRecordBlock()	32
3.4.3.10 getRecordCount()	33
3.4.3.11 makeRecordOffsets()	33
3.4.3.12 rewriteSSFile()	33
3.4.3.13 sKeyStateBuilder()	34
3.4.3.14 test()	34
3.4.3.15 writeBlocks()	35
3.4.3.16 writeToTxt()	35
3.5 Truncate Class Reference	36
3.5.1 Detailed Description	36
3.5.2 Constructor & Destructor Documentation	36
3.5.2.1 Truncate() [1/2]	36
3.5.2.2 Truncate() [2/2]	36
3.5.3 Member Function Documentation	37
3.5.3.1 modifyString()	37
3.5.3.2 truncatedString()	37

4 File Documentation	39
4.1 Doxygen/Input/Block.cpp File Reference	39
4.1.1 Function Documentation	40
4.1.1.1 binarySearch()	41
4.1.1.2 convertIntArrToStrArr()	41
4.1.1.3 convertStrArrToIntArr()	42
4.1.2 Variable Documentation	42
4.1.2.1 NULL_INT	42
4.1.2.2 null_str	42
4.2 Block.cpp	43
4.3 Doxygen/Input/Block.h File Reference	47
4.4 Block.h	48
4.5 Doxygen/Input/grid.cpp File Reference	49
4.6 grid.cpp	49
4.7 Doxygen/Input/Header.cpp File Reference	51
4.7.1 Variable Documentation	52
4.7.1.1 BLOCKFILLCOUNT	52
4.7.1.2 BLOCKLENGTH	52
4.7.1.3 DATAFILENAME	52
4.7.1.4 DEBUG	52
4.7.1.5 FILLPERCENT	53
4.7.1.6 HEADERENDSTRING	53
4.7.1.7 RBNLENGTH	53
4.7.1.8 RECORDSPERBLOCK	53
4.7.1.9 ZIPLength	53
4.8 Header.cpp	53
4.9 Doxygen/Input/main.cpp File Reference	54
4.9.1 Function Documentation	54
4.9.1.1 blockTester()	54
4.9.1.2 main()	55
4.9.1.3 nullblockTester()	56
4.9.1.4 recordTester()	56
4.9.1.5 SSDeleteAndAddRecordTester()	57
4.9.1.6 truncateTester()	57
4.10 main.cpp	58
4.11 Doxygen/Input/Record.cpp File Reference	60
4.12 Record.cpp	61
4.13 Doxygen/Input/Record.h File Reference	66
4.14 Record.h	67
4.15 Doxygen/Input/Record_Test_Driver.cpp File Reference	67
4.15.1 Enumeration Type Documentation	68
4.15.1.1 Field	68

4.15.2 Function Documentation	69
4.15.2.1 main()	69
4.16 Record_Test_Driver.cpp	69
4.17 Doxygen/Input/SequenceSet.cpp File Reference	70
4.17.1 Function Documentation	71
4.17.1.1 binarySearchSS()	71
4.18 SequenceSet.cpp	72
4.19 Doxygen/Input/SequenceSet.h File Reference	82
4.20 SequenceSet.h	83
4.21 Doxygen/Input/Truncate.cpp File Reference	83
4.21.1 Detailed Description	84
4.22 Truncate.cpp	85
4.23 Doxygen/Input/Truncate.h File Reference	85
4.24 Truncate.h	86
Index	87

Chapter 1

Class Index

1.1 Class List

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

Block	5
Grid		
Grid class	15
Record	19
SequenceSet	25
Truncate	36

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Doxygen/Input/Block.cpp	39
Doxygen/Input/Block.h	47
Doxygen/Input/grid.cpp	49
Doxygen/Input/Header.cpp	51
Doxygen/Input/main.cpp	54
Doxygen/Input/Record.cpp	60
Doxygen/Input/Record.h	66
Doxygen/Input/Record_Test_Driver.cpp	67
Doxygen/Input/SequenceSet.cpp	70
Doxygen/Input/SequenceSet.h	82
Doxygen/Input/Truncate.cpp	83
Doxygen/Input/Truncate.h	85

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

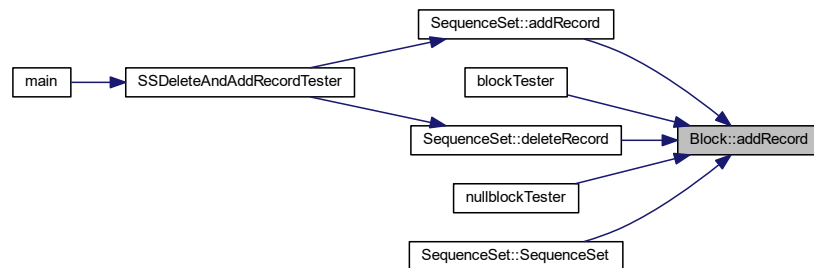
Primary key

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

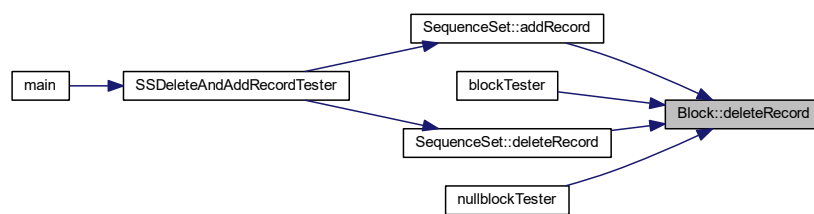
Primary key

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:



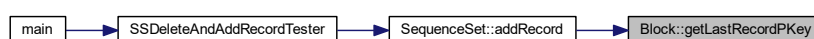
3.1.3.4 getLastRecordPKey()

```
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:



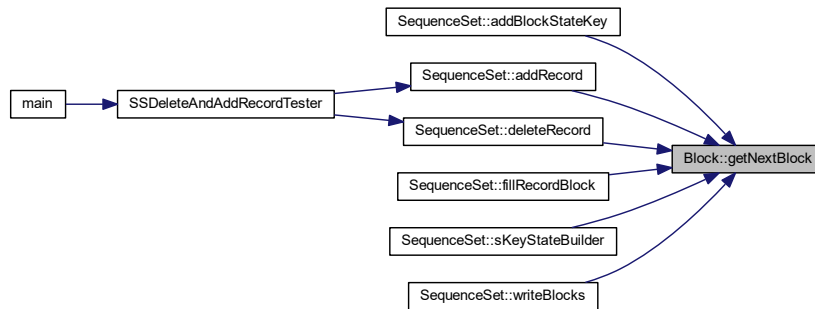
3.1.3.5 getNextBlock()

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

Gets pointer of next block.

Definition at line 192 of file [Block.cpp](#).

Here is the caller graph for this function:



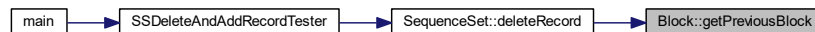
3.1.3.6 getPreviousBlock()

```
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](#).

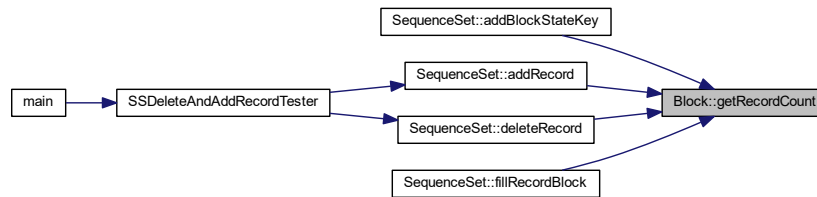
3.1.3.8 getRecordCount()

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

Gets the record count.

Definition at line 220 of file [Block.cpp](#).

Here is the caller graph for this function:



3.1.3.9 getRecords()

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

Gets zip codes for the sequence set.

Precondition

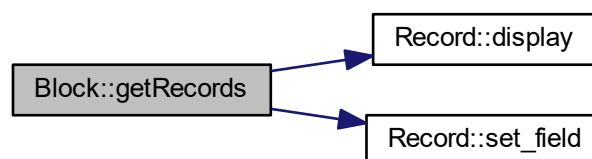
[Block](#)

Postcondition

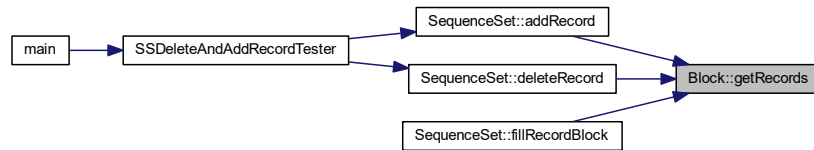
Returns records in a block

Definition at line 289 of file [Block.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.3.10 search()

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

Searches for record.

Precondition

Primary key

Postcondition

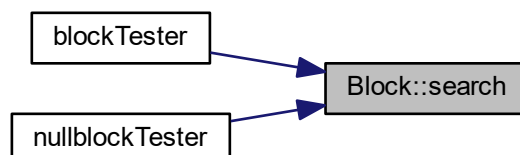
Returns the record or 0 if the record is not found Searches for primary key

Definition at line 185 of file [Block.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



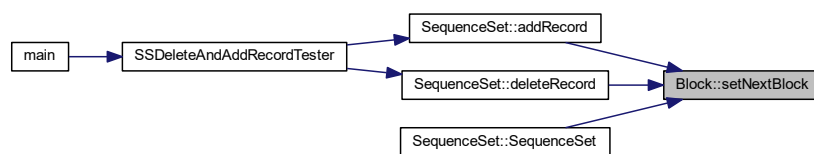
3.1.3.11 setNextBlock()

```
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:



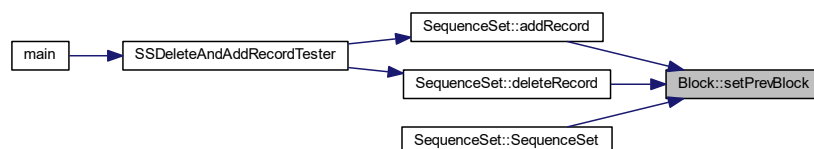
3.1.3.12 setPrevBlock()

```
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

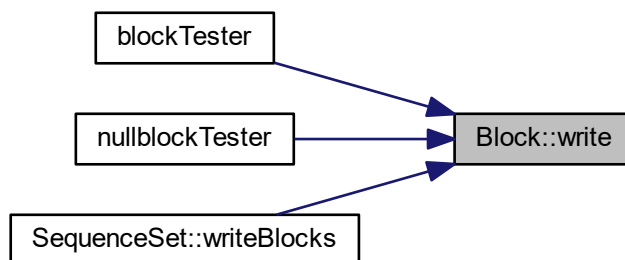
A block

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

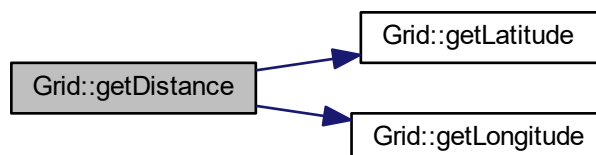
grid object must be provided

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

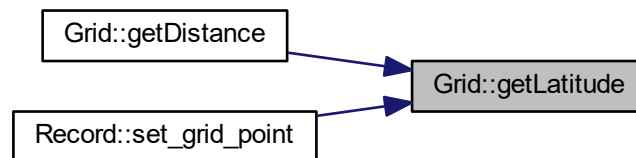
none

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

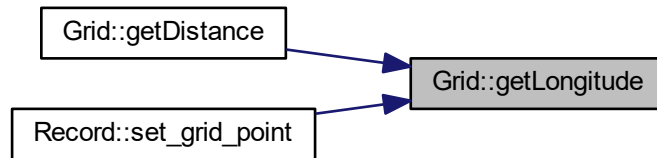
none

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

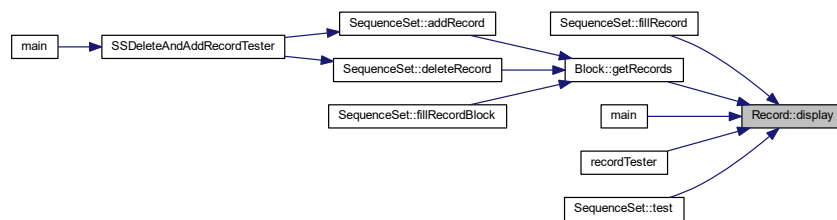
None

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

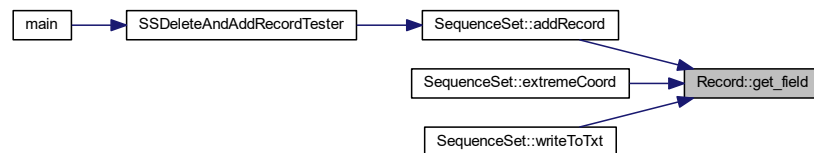
Provided string must match the name of a field in the record

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

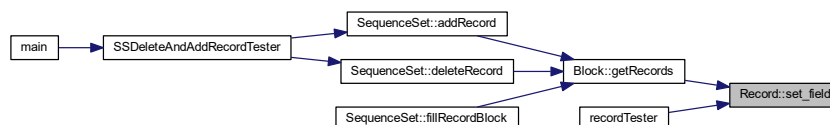
First provided string must match the name of a field in the record Second provided string must be the appropriate length for the field

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:



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

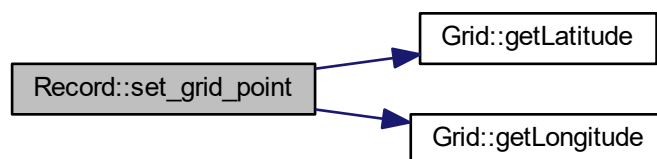
A grid point of type [Grid](#)

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](#).

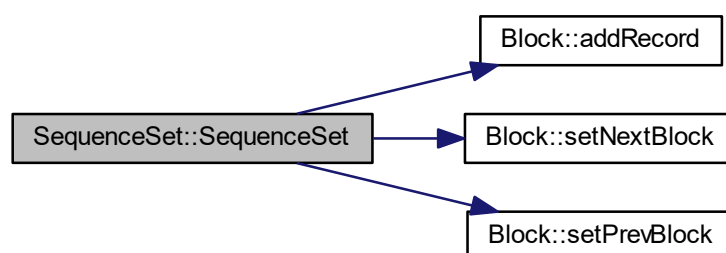
3.4.2 Constructor & Destructor Documentation

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

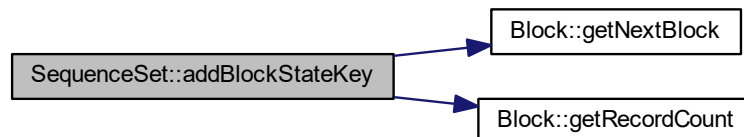
[Block](#) ID is less than block count

Postcondition

*****add post

Definition at line [355](#) of file [SequenceSet.cpp](#).

Here is the call graph for this function:

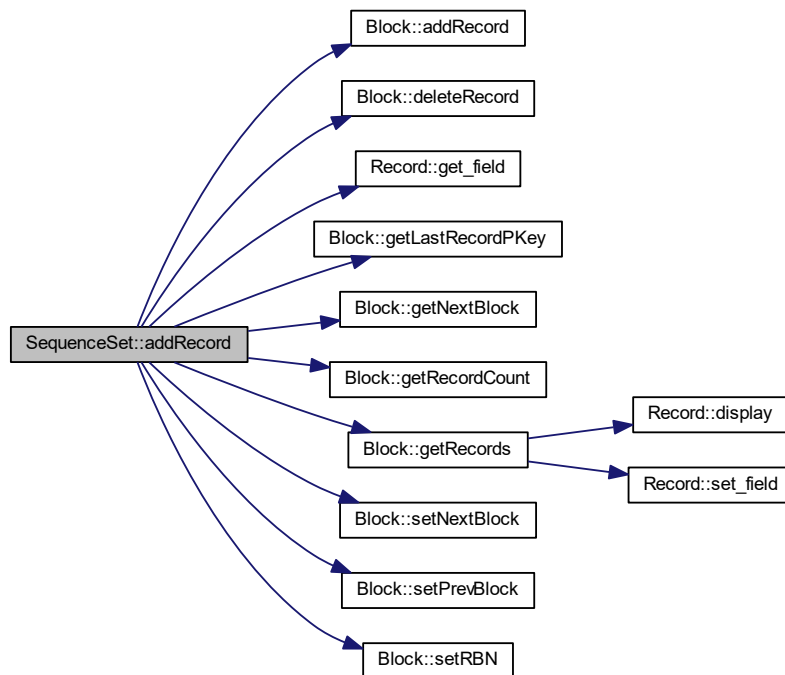


3.4.3.2 addRecord()

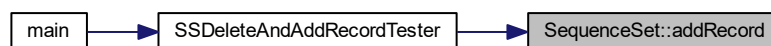
```
void SequenceSet::addRecord (  
    Record record )
```

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()

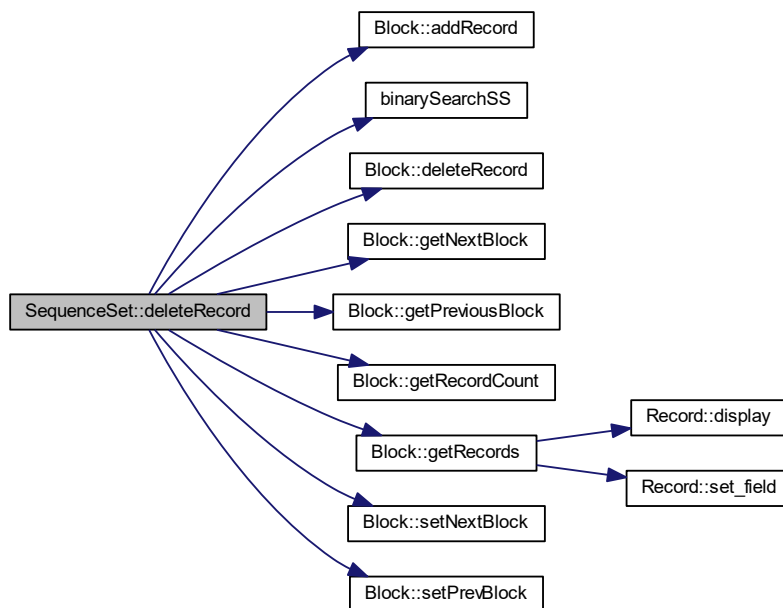
```

bool SequenceSet::deleteRecord (
    int pKey )

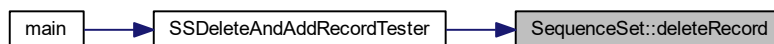
```

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

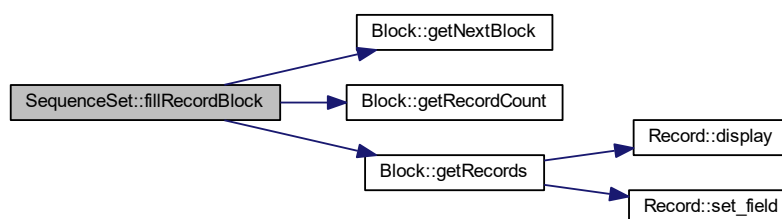
blockID must be less than the block count

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

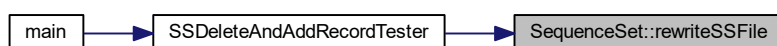
Definition at line 179 of file [SequenceSet.cpp](#).

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

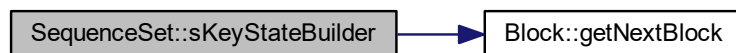
None

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

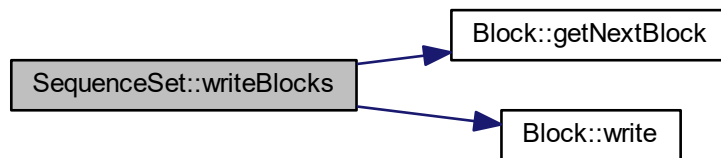
None

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:

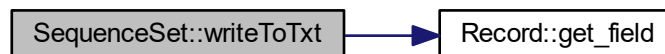


3.4.3.16 writeToTxt()

```
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](#)

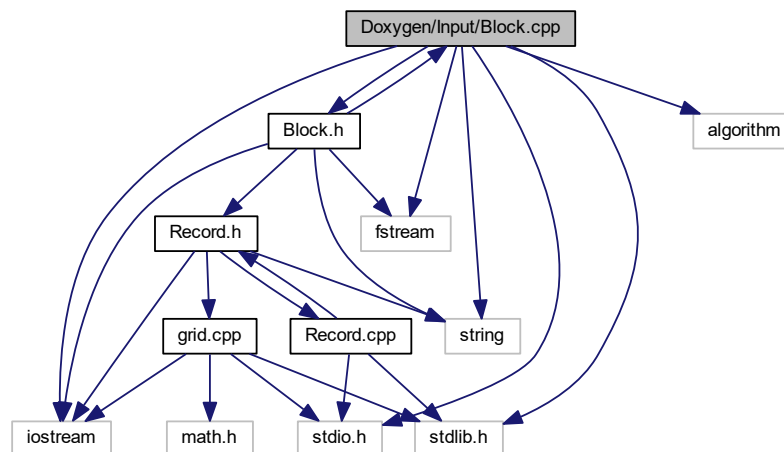
Chapter 4

File Documentation

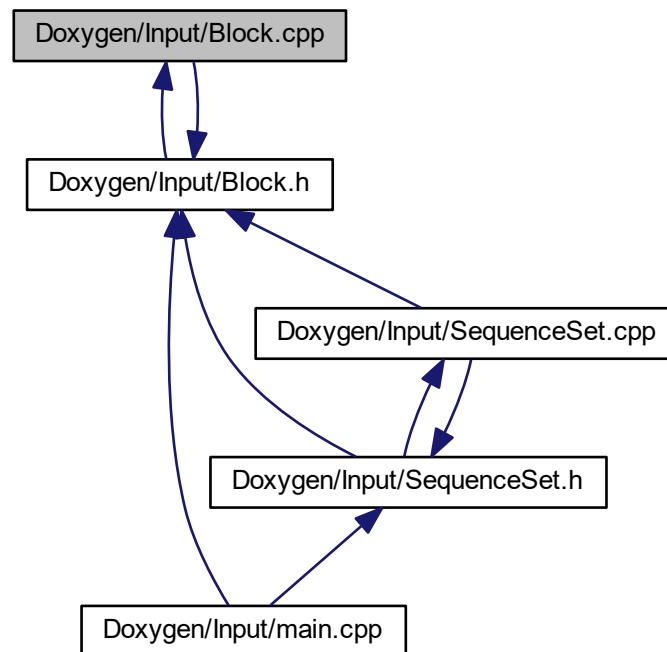
4.1 Doxygen/Input/Block.cpp File Reference

```
#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

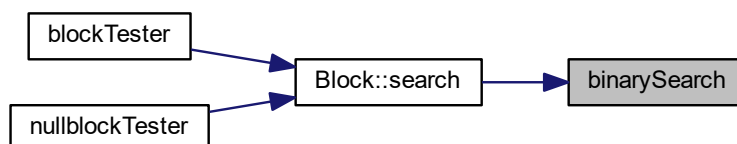
Primary key

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

```

00001
00022 #include "Block.h"
00023 #include <iostream>
00024 #include <fstream>
00025 #include <string>
00026 #include <stdio.h>
00027 #include <stdlib.h>
00028 #include <algorithm>
00029
00030 using namespace std;
00031
00032 //prototype for binary search
00033 int binarySearch(const string[], string,int);
00034 void convertStrArrToIntArr(const string[], int[], int);
00035 void convertIntArrToStrArr(string [], int [], int );
00036
00037 const string null_str = "";
00038 const int NULL_INT = 1000000;
00039
00040 Block::Block()
00041 {
00042     isEmpty = true;
00043     relativeBlockNumber = 0;
00044     recordCount = 0;
00045     for(int i = 0; i < RECORDSPERBLOCK; i++){
00046         records[i] = "";
00047     }
00048
00049     nextBlock = nullptr;
00050     previousBlock = nullptr;
00051
00052     if(DEBUG) {cout << "Made an empty block.\n";}
00053 }
00054
00055 Block::Block(unsigned long long _RBN)
00056 {
00057     isEmpty = true;
00058     relativeBlockNumber = _RBN;
00059     recordCount = 0;
00060     for(int i = 0; i < RECORDSPERBLOCK; i++){
00061         records[i] = "";
00062     }
00063
00064     nextBlock = nullptr;
00065     previousBlock = nullptr;
00066
00067     if(DEBUG) {cout << "Made an empty block.\n";}
00068 }
00069
00070 string Block::blockData(){
00071     string returnString = "";
00072     returnString += relativeBlockNumber;
00073     for(int i = 0; i < recordCount; i++){
00074         returnString += " ";
00075         returnString += records[i];
00076     }
00077     return returnString;
00078 }
00079
00080 Block::Block(string _blockData)
00081 {
00082     if(DEBUG) {cout << "Making a block with \"" << _blockData << "\".\n";}
00083
00084     isEmpty = false;
00085     relativeBlockNumber = 0;
00086     recordCount = 0;
00087
00088     //set the primary keys of each record
00089     string tempStr = "";
00090     int recordNumber = 0;
00091     int j = 0; //pointer to track the position in the string
00092     while( j < _blockData.length() && j < BLOCKFILLCOUNT*ZIPLNGTH)
00093     {
00094         for(int i = 0; i < ZIPLNGTH; i++) //for each element of the pKey
00095         {
00096             if( _blockData[j] >= '0' && _blockData[j] <= '9' )
00097             {
00098                 tempStr += _blockData[j]; //if the element is numeric, store the value
00099             }
00100             j++; //increment the pointer
00101         }
00102         records[recordNumber] = tempStr; //store the pKey in the class
00103         tempStr = ""; //clear the temp string
00104         if(records[recordNumber] != ""){
00105             recordCount++; //update the number of records in the block

```

```

00106         recordNumber++; //increment the record number
00107     }
00108 }
00109
00110 if(DEBUG) {cout << "Elements of Constructed block " << relativeBlockNumber << ": \"";
00111             for(int i = 0; i < RECORDSPERBLOCK; i++){cout << records[i] << " ";}
00112             cout << "\".\n";}
00113
00114 nextBlock = nullptr;
00115 previousBlock = nullptr;
00116 }
00117
00118 Block::Block(string _blockData[RECORDSPERBLOCK])
00119 {
00120     if(DEBUG) {cout << "Making a block with \"";
00121                 for(int i = 0; i < BLOCKFILLCOUNT; i++){cout << _blockData[i];}
00122                 cout << "\".\n";}
00123
00124     isEmpty = false;
00125     relativeBlockNumber = 0;
00126     recordCount = 0;
00127
00128     //set the primary keys of each record
00129     for(int i = 0; i < BLOCKFILLCOUNT; i++)
00130     {
00131         records[i] = _blockData[i];
00132         if(records[i] != ""){
00133             recordCount++;
00134         }
00135     }
00136
00137     if(DEBUG) {cout << "Elements of Constructed block " << relativeBlockNumber << ": \"";
00138                 for(int i = 0; i < BLOCKFILLCOUNT; i++){cout << records[i] << " ";}
00139                 cout << "\".\n";}
00140
00141     nextBlock = nullptr;
00142     previousBlock = nullptr;
00143 }
00144
00145
00146 void Block::write(string _fileName)
00147 {
00148     ofstream file;
00149     file.open(_fileName, ios_base::app); if(DEBUG) {cout << "Writing block number " <<
relativeBlockNumber << " to a " << _fileName << "\".\n";}
00150
00151     file.seekp(relativeBlockNumber * BLOCKLENGTH);
00152
00153     if(DEBUG){
00154         cout << relativeBlockNumber << ": ";
00155         for(int i = 0; i < recordCount; i++){
00156             cout << records[i] << " ";
00157         }
00158         cout << endl;
00159     }
00160
00161     file << "RBN " << relativeBlockNumber << ": ";
00162     if(DEBUG){cout << "The file should read: \"";}
00163     if(DEBUG){cout << "RBN " << relativeBlockNumber << ": ";}
00164     for(int i = 0; i < recordCount; i++){
00165         string record = records[i];
00166         for(int j = ZIPLength - record.length(); j > 0; j--){
00167             file << " ";
00168             if(DEBUG){cout << " ";}
00169         }
00170
00171         file << record;
00172         if(DEBUG){cout << record;}
00173     }
00174     for(int i = RECORDSPERBLOCK - recordCount; i > 0; i--){
00175         for(int j = 0; j < ZIPLength; j++){
00176             file << " ";
00177             if(DEBUG){cout << " ";}
00178         }
00179     }
00180     file << "\".\n";
00181     if(DEBUG){cout << "\".\n";}
00182     file.close();
00183 }
00184
00185 int Block::search(string pKey)
00186 {
00187     if(DEBUG) {cout << "Searching for " << pKey << " in this block\n";}
00188
00189     return binarySearch(records, pKey, RECORDSPERBLOCK );
00190 }
00191

```

```

00192 Block * Block::getNextBlock()
00193 {
00194     return nextBlock;
00195 }
00196     if(DEBUG) {cout << "Pointer to the next block has been returned.\n";}
00197 }
00198
00199 Block * Block::getPreviousBlock()
00200 {
00201     return previousBlock;
00202 }
00203     if(DEBUG) {cout << "Pointer to the previous block has been returned.\n";}
00204 }
00205
00206 void Block::setNextBlock( Block * nextBlockPtr )
00207 {
00208     nextBlock = nextBlockPtr;
00209 }
00210     if(DEBUG) {cout << "Pointer to the next block has been set.\n";}
00211 }
00212
00213 void Block::setPrevBlock( Block * previousBlockPtr )
00214 {
00215     previousBlock = previousBlockPtr;
00216 }
00217     if(DEBUG) {cout << "Pointer to the previous block has been set.\n";}
00218 }
00219
00220 int Block::getRecordCount()
00221 {
00222     return recordCount;
00223 }
00224
00225 int Block::getLastRecordPKey()
00226 {
00227     if(DEBUG) {cout << "Getting last record of the block\n";}
00228     return stoi( records[ recordCount - 1 ] );
00229 }
00230
00231 bool Block::deleteRecord(string pKey)
00232 {
00233     if(DEBUG) {cout << "Deleting record " << pKey << " in block " << relativeBlockNumber << "\n";}
00234     if(DEBUG) {cout << "Elements of Constructed block before deleting record: \"\" ;
00235         for(int i = 0; i < RECORDSPERBLOCK; i++){cout << records[i];}
00236         cout << "\".\n";}
00237
00238     int position = this -> search(pKey); //get the position of the record to be deleted
00239
00240     if ( position != -1 )
00241     {
00242         records[position] = ""; //delete the record
00243         recordCount--; //decrement record count
00244         if(DEBUG) {cout << "Elements of Constructed block after deleting record: \"\" ;
00245             for(int i = 0; i < RECORDSPERBLOCK; i++){if(records[i] == null_str){cout << "null";}else{cout <<
00246                 records[i];}}
00247             cout << "\".\n";}
00248             sortRecord(); //sort the record
00249             if(DEBUG) {cout << "Elements of Constructed block after sorting record: \"\" ;
00250                 for(int i = 0; i < RECORDSPERBLOCK; i++){if(records[i] == null_str){cout << "null";}else{cout <<
00251                     records[i];}}
00252                 cout << "\".\n";}
00253             return true;
00254         }
00255         else
00256         {
00257             if(DEBUG) {cout << "Record not found in block. Could not delete" << "\".\n";}
00258             return false;
00259         }
00260     }
00261
00262 bool Block::addRecord(string pKey)
00263 {
00264     if(DEBUG) {cout << "Adding a record to " << relativeBlockNumber << "\n";}
00265     if(DEBUG) {cout << "Elements of Constructed block before adding record: \"\" ;
00266         for(int i = 0; i < RECORDSPERBLOCK; i++){cout << records[i];}
00267         cout << "\".\n";}
00268
00269     for(int i = 0; i < RECORDSPERBLOCK; i++) //go through the block to see if there is empty record
00270     {
00271         if( records[i] == null_str) //if there is an empty record
00272         {
00273             records[i] = pKey; //fill the record with the pKey
00274             recordCount++; //increment record count
00275             if(DEBUG) {cout << "Elements of Constructed block after adding record: \"\" ;
00276                 for(int i = 0; i < RECORDSPERBLOCK; i++){cout << records[i];}
00277                 cout << "\".\n";}
00278             sortRecord(); //sort the record

```

```

00277         if(DEBUG) {cout << "Elements of Constructed block after sorting record: \"\" ;
00278                     for(int i = 0; i < RECORDSPERBLOCK; i++){cout << records[i];}
00279                     cout << "\".\n\";}}
00280         return true;
00281     }
00282 }
00283
00284 if(DEBUG) {cout << "Block Full. Could not add record." << "\".\n\";}}
00285
00286 return false;
00287 }
00288
00289 void Block::getRecords(Record block[])
00290 {
00291     if(DEBUG) {cout << "Setting record zips\n\";}}
00292
00293     for(auto i = 0; i < RECORDSPERBLOCK; i++){
00294         block[i].set_field("ZIP", records[i]);
00295         if(DEBUG){
00296             cout<< "Block["<i>i</i>"]: " << endl;
00297             block[i].display();
00298         }
00299     }
00300 }
00301
00302 unsigned long long Block::getRBN(){
00303     return relativeBlockNumber;
00304 }
00305
00306 void Block::setRBN(unsigned long long RBN){
00307     relativeBlockNumber = RBN;
00308 }
00309
00310 void Block::sortRecord()
00311 {
00312     if(DEBUG) {cout << "Sorting the records in the block.\n\";}}
00313
00314     int int_records_array[RECORDSPERBLOCK]; //to convert the string of records to integers
00315     convertStrArrToIntArr(records, int_records_array, RECORDSPERBLOCK);
00316
00317     int n = sizeof(int_records_array)/sizeof(int_records_array[0]);
00318     sort(int_records_array, int_records_array+n);
00319
00320     //convert back to strings and store in records array of string
00321     convertIntArrToStrArr(records, int_records_array, RECORDSPERBLOCK);
00322 }
00323
00324 int binarySearch(const string arr[], string x, int n)
00325 {
00326     int int_arr[n];
00327     int int_string;
00328
00329     //convert the records (array of strings) to array of int
00330     for (int i = 0; i < n; i++)
00331     {
00332         if(arr[i] != null_str)
00333             int_arr[i] = stoi(arr[i]);
00334     }
00335
00336     //convert string to find to int
00337     int_string = stoi(x);
00338
00339     int l = 0 ;
00340     int r = n - 1;
00341     while (l <= r)
00342     {
00343         int m = l + (r - l) / 2;
00344         if(DEBUG) {cout << "mid: " << m << endl;}}
00345
00346         if(DEBUG) {cout << "comparing " << int_string << " and " << int_arr[m] << endl;}}
00347
00348         if ( int_arr[m] == int_string ){
00349             if(DEBUG) {cout << "record found" << endl;}}
00350             return m;
00351         }
00352
00353         // If x is greater, ignore left half
00354         if ( int_arr[m] < int_string ){
00355             l = m + 1;
00356             if(DEBUG) {cout << "new l: " << l << endl;}}
00357         }
00358
00359         // If x is smaller, ignore right half
00360         else{
00361             r = m - 1;
00362             if(DEBUG) {cout << "new r: " << l << endl;}}
00363         }
00364     }

```

```

00368     }
00369
00370     return -1;
00371 }
00372
00377 void convertStrArrToIntArr(const string strArr[], int intArr[], int ArrLength)
00378 {
00379     //convert the records (array of strings) to array of int
00380     for (int i = 0; i < ArrLength; i++)
00381     {
00382         if(strArr[i] == null_str) //if the record is null
00383             intArr[i] = NULL_INT;
00384         else
00385             intArr[i] = stoi(strArr[i]);
00386     }
00387 }
00388
00393 void convertIntArrToStrArr(string strArr[], int intArr[], int ArrLength)
00394 {
00395     //convert the records (array of strings) to array of int
00396     for (int i = 0; i < ArrLength; i++)
00397     {
00398         if(intArr[i] == NULL_INT)//if the record is null
00399             strArr[i] = null_str;
00400         else
00401             strArr[i] = to_string(intArr[i]);
00402     }
00403 }

```

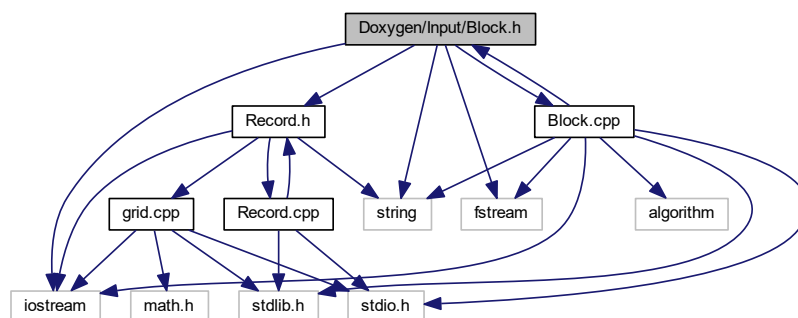
4.3 Doxygen/Input/Block.h File Reference

```

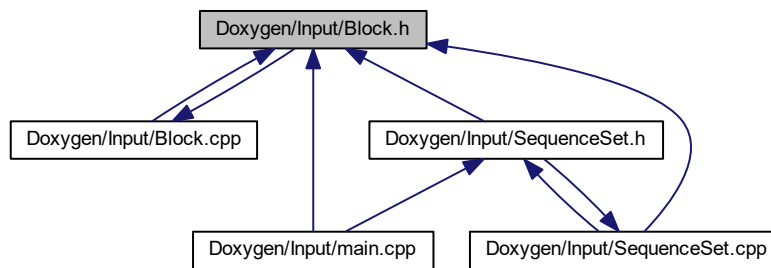
#include <iostream>
#include <string>
#include <fstream>
#include "Record.h"
#include "Block.cpp"

```

Include dependency graph for Block.h:



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



Classes

- class [Block](#)

4.4 Block.h

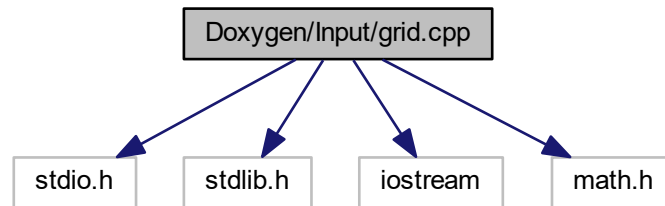
```

00001
00021 #ifndef BLOCK_H
00022 #define BLOCK_H
00023
00024 #include <iostream>
00025 #include <string>
00026 #include <fstream>
00027 #include "Record.h"
00028 using namespace std;
00029
00030 class Block
00031 {
00032 public:
00033     Block();
00042     Block(unsigned long long _RBN);
00047     Block(string[]);
00052     Block(string);
00057     void write(string);
00062     int search(string pKey);
00063     Block * getNextBlock();
00064     Block * getPreviousBlock();
00065     void setNextBlock( Block * nextBlockPtr );
00066     void setPrevBlock( Block * previousBlockPtr );
00067     int getRecordCount();
00068     int getLastRecordPKey();
00073     bool deleteRecord(string pKey);
00078     bool addRecord(string pKey);
00083     void getRecords(Record block[]);
00084     string blockData();
00086     unsigned long long getRBN();
00087     void setRBN(unsigned long long);
00088 private:
00089     void sortRecord();
00090     bool isEmpty();
00091     unsigned long long relativeBlockNumber;
00092     int recordCount;
00093     string records[RECORDSPERBLOCK];
00094     Block * nextBlock;
00095     Block * previousBlock;
00096 };
00097
00098 #include "Block.cpp"
00099
00100 #endif
  
```

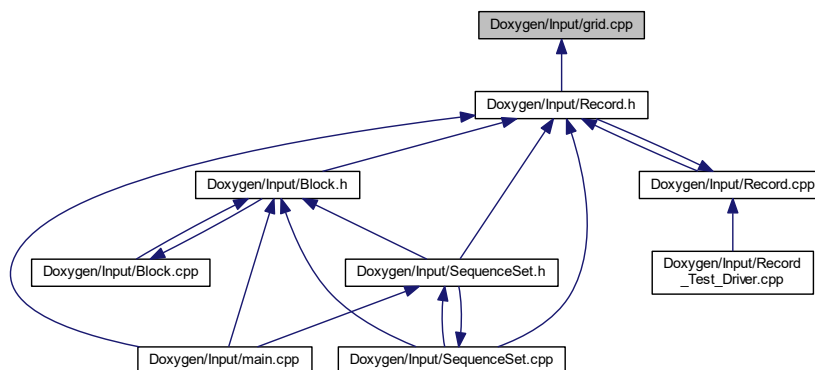
4.5 Doxygen/Input/grid.cpp File Reference

```
#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.

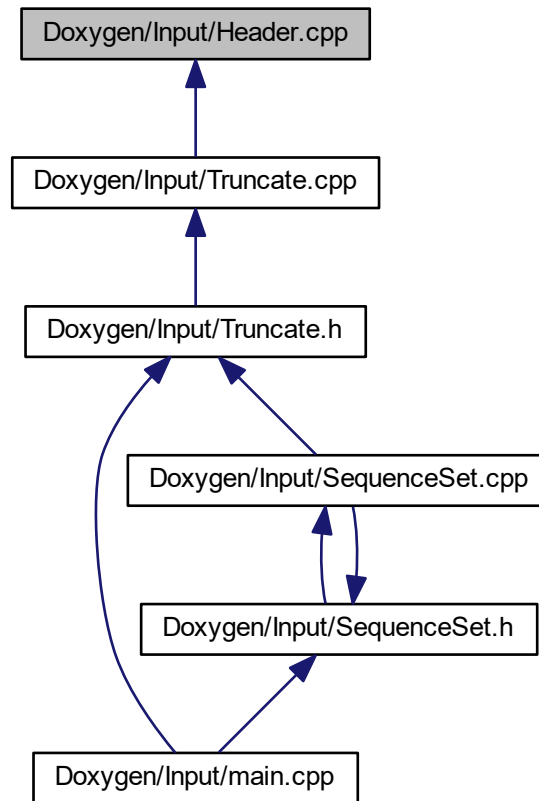
4.6 grid.cpp

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

```
00019 //const bool DEBUG = true;
00020
00030 class Grid {
00031     private:
00032         float latitude;
00033         float longitude;
00035     public:
00036         Grid();
00037         Grid(float, float);
00038         void setLatitude(float);
00039         void setLongitude(float);
00040         void setLatitude(string);
00041         void setLongitude(string);
00042         float getLatitude();
00043         float getLongitude();
00044         float getDistance(Grid);
00045 };
00046
00051 Grid::Grid(){
00052     latitude = 0;
00053     longitude = 0;
00054 }
00055
00060 Grid::Grid(float _latitude, float _longitude){
00061     latitude = _latitude;
00062     longitude = _longitude;
00063 }
00064
00069 void Grid::setLatitude(float _latitude){
00070     latitude = _latitude;
00071 }
00072
00077 void Grid::setLatitude(string _latitude){
00078     setLatitude(stof(_latitude));
00079 }
00080
00085 void Grid::setLongitude(float _longitude){
00086     longitude = _longitude;
00087 }
00088
00093 void Grid::setLongitude(string _longitude){
00094     setLongitude(stof(_longitude));
00095 }
00096
00101 float Grid::getLatitude(){
00102     return latitude;
00103 }
00104
00109 float Grid::getLongitude(){
00110     return longitude;
00111 }
00112
00117 float Grid::getDistance(Grid _grid){
00118     float distance = pow(latitude - _grid.getLatitude(),2) + pow(longitude - _grid.getLongitude(),2);
00119     distance = sqrt(distance);
00120     return distance;
00121 }
```


4.7 Doxygen/Input/Header.cpp File Reference

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



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.

4.7.1.5 FILLPERCENT

```
const double FILLPERCENT = 75
```

Max length of the RBN code in digits.

4.7.1.6 HEADERENDSTRING

[illegible]

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

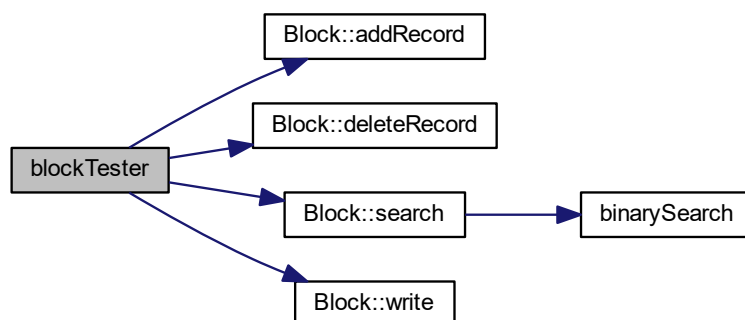
```
const int ZIPLength = 6
```

Max length of the zip code in digits.

4.8 Header.cpp

```
00001 extern const bool DEBUG = false;
00002 extern const int RECORDSPERBLOCK = 4;
00003 extern const int ZIPLength = 6;
00004 extern const int RBNLength = 8;
00005 extern const int BLOCKLength = RBNLength + RECORDSPERBLOCK * ZIPLength;
00006 extern const double FILLPERCENT = 75;
00007 extern const int BLOCKFILLCOUNT = RECORDSPERBLOCK * (FILLPERCENT/100);
00008 extern const string HEADERENDSTRING =
    "123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890";
00009 extern const string DATAFILENAME = "us_postal_codes.txt";
```


Here is the call graph for this function:

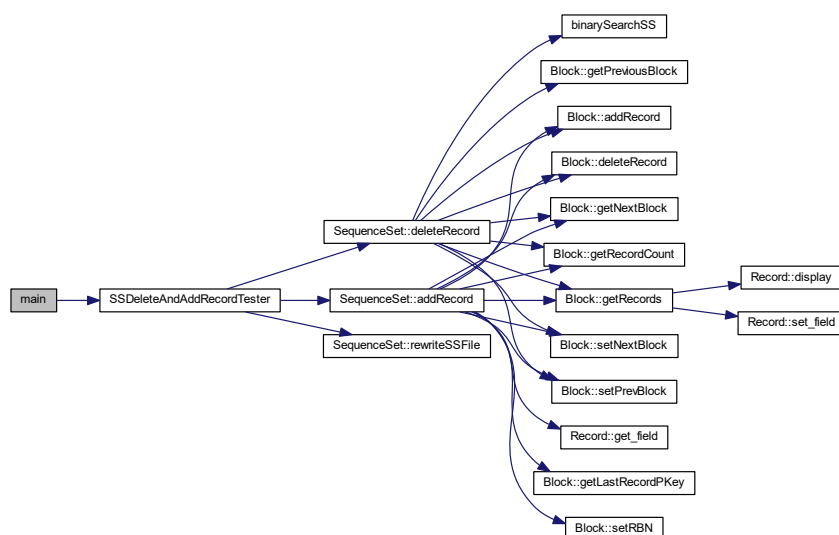


4.9.1.2 main()

```
int main ( )
```

Definition at line 17 of file [main.cpp](#).

Here is the call graph for this function:

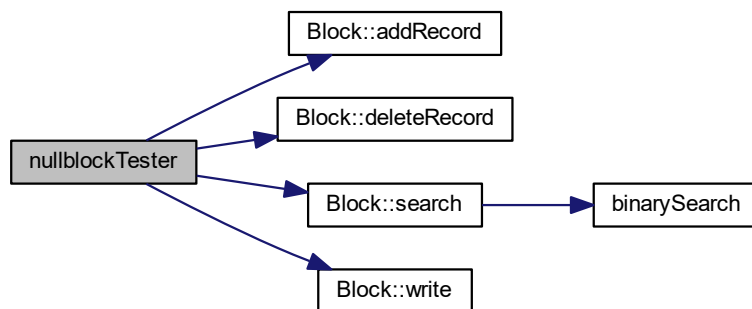


4.9.1.3 nullblockTester()

```
void nullblockTester ( )
```

Definition at line 65 of file [main.cpp](#).

Here is the call graph for this function:

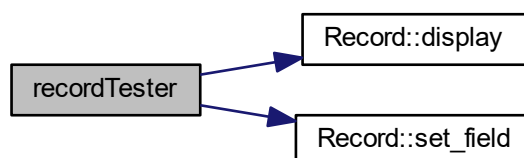


4.9.1.4 recordTester()

```
void recordTester ( )
```

Definition at line 141 of file [main.cpp](#).

Here is the call graph for this function:

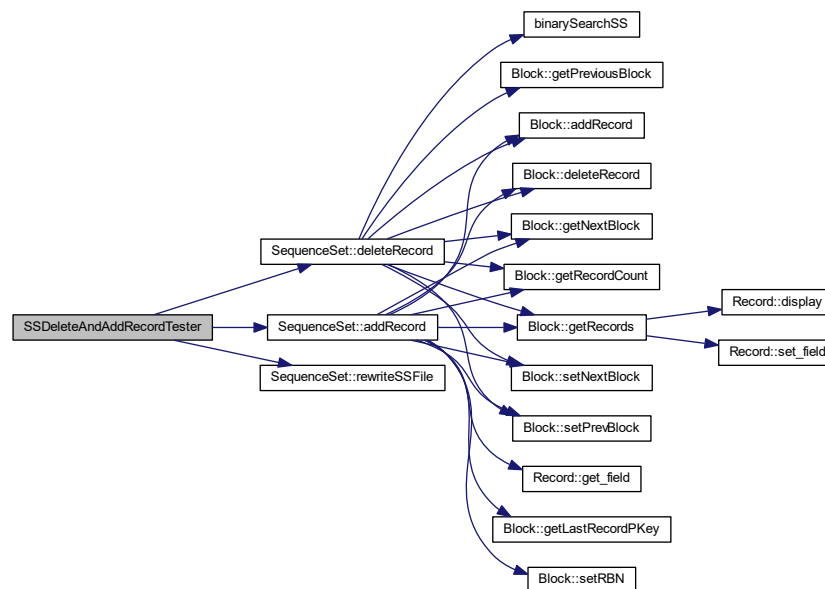


4.9.1.5 SSDeleteAndAddRecordTester()

```
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:



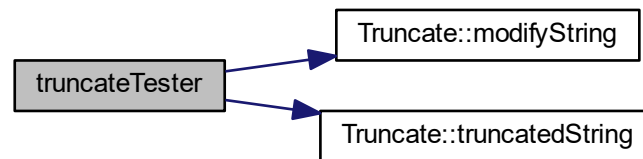
4.9.1.6 truncateTester()

```
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

```

00001 #include <iostream>
00002 #include "Truncate.h"
00003 #include "Record.h"
00004 #include "Block.h"
00005 #include "SequenceSet.h"
00006 #include <string>
00007 #include <fstream>
00008
00009 using namespace std;
00010
00011 void truncateTester();
00012 void recordTester();
00013 void blockTester();
00014 void nullblockTester();
00015 void SSDeleteAndAddRecordTester();
00016
00017 int main(){
00018     //nullblockTester();
00019     //recordTester();
00020     SSDeleteAndAddRecordTester();
00021     //SequenceSet SSClass; SSClass.test();
00022     return 0;
00023 }
00024
00025 void SSDeleteAndAddRecordTester()
00026 {
00027     SequenceSet SSClass;
00028
00029     SSClass.deleteRecord(1008);
00030     SSClass.deleteRecord(1003);
00031     SSClass.deleteRecord(1004);
00032
00033     string zip = "563";
00034     string place = "Little Falls";
00035     string state = "MN";
00036     string county = "Morrison";
00037     string longitude = "-74.25";
00038     string latitude = "79.72";
00039     Record testRecord(zip, place, state, county, longitude, latitude);
00040     SSClass.addRecord(testRecord);
00041
00042     zip = "1024";
00043     Record testRecord2(zip, place, state, county, longitude, latitude);
00044     SSClass.addRecord(testRecord2);
00045
00046     zip = "1025";
00047     Record testRecord3(zip, place, state, county, longitude, latitude);
00048     SSClass.addRecord(testRecord3);
00049
00050     zip = "1051";
00051     Record testRecord4(zip, place, state, county, longitude, latitude);
00052     SSClass.addRecord(testRecord4);
00053
00054     zip = "1052";
00055     Record testRecord5(zip, place, state, county, longitude, latitude);
00056     SSClass.addRecord(testRecord5);
00057
00058     zip = "300";
00059     Record testRecord6(zip, place, state, county, longitude, latitude);
  
```



```

00060     SSClass.addRecord(testRecord6);
00061
00062     SSClass.rewriteSSFile();
00063 }
00064
00065 void nullblockTester(){
00066     Block * aBlock;
00067     ofstream sequenceSetFile;
00068     string fileName = "Sequence_Set.txt";
00069     sequenceSetFile.open(fileName);
00070     sequenceSetFile << "Hello File\n";
00071     sequenceSetFile.close();
00072
00073     string records[4] = {"501", "544", "1001", ""};
00074     string blockInfo = "    501    544    1001    1002";
00075
00076     //test block constructor
00077     Block anotherBlock(blockInfo);
00078     anotherBlock.write(fileName);
00079
00080     //test block search method
00081     string recordTest = "1002";
00082     aBlock = new Block(1);
00083     cout << "Return 1 if the record was found: " << aBlock->search( recordTest ) << endl;
00084
00085     recordTest = "103";
00086     aBlock->addRecord(recordTest);
00087
00088     recordTest = "103";
00089     aBlock->addRecord(recordTest);
00090
00091     recordTest = "544";
00092     aBlock->deleteRecord(recordTest);
00093
00094     recordTest = "514";
00095     aBlock->deleteRecord(recordTest);
00096 }
00097
00098 void blockTester(){
00099     Block aBlock;
00100     ofstream sequenceSetFile;
00101     string fileName = "Sequence_Set.txt";
00102     sequenceSetFile.open(fileName);
00103     sequenceSetFile << "Hello File\n";
00104     sequenceSetFile.close();
00105
00106     string records[4] = {"501", "544", "1001", ""};
00107     string blockInfo = "    501    544    1001    1002";
00108
00109     //test block constructor
00110     Block anotherBlock(blockInfo);
00111     anotherBlock.write(fileName);
00112
00113     //test block search method
00114     string recordTest = "1002";
00115     cout << "Return 1 if the record was found: " << anotherBlock.search( recordTest ) << endl;
00116
00117     recordTest = "103";
00118     anotherBlock.addRecord(recordTest);
00119
00120     recordTest = "103";
00121     anotherBlock.addRecord(recordTest);
00122
00123     recordTest = "544";
00124     anotherBlock.deleteRecord(recordTest);
00125
00126     recordTest = "514";
00127     anotherBlock.deleteRecord(recordTest);
00128 }
00129
00130 void truncateTester(){
00131     Truncate t;
00132     Truncate t2(5);
00133     string str = "123456789AB";
00134
00135     cout << endl << "The String is " << str;
00136     cout << endl << "The String AS it is modified is " << t.modifyString(str);
00137     cout << endl << "The String IF it was modified is " << t2.truncatedString(str);
00138     cout << endl << "The String is " << str << endl;
00139 }
00140
00141 void recordTester(){
00142     //test default constructor
00143     Record testRecord;
00144     cout << "Default constructor record (should be empty):";
00145     testRecord.display();
00146     cout << endl;

```

```

00147
00148 //test fill record
00149 string zip = "56345";
00150 string place = "Little Falls";
00151 string state = "Minnesota";
00152 string county = "Morrison";
00153 string longitude = "-74.25";
00154 string latitude = "79.72";
00155
00156 cout << "Fill Record with : " << zip << " " << place << " " << state << " " << county << " " << longitude << "
" << latitude;

00157
00158 testRecord.set_field( "z", zip );
00159 testRecord.set_field( "place", place );
00160 testRecord.set_field( "STATE", state );
00161 testRecord.set_field( "c", county );
00162 testRecord.set_field( "long", longitude );
00163 testRecord.set_field( "lat", latitude );
00164
00165 testRecord.display();
00166 cout << endl;
00167
00168 //test constructor 2
00169 float longitude_float = 74.25;
00170 float latitude_float = 79.72;
00171
00172 Record testRecord2(zip, place, state, county, longitude, latitude);
00173
00174 cout << "Constructor2 record (record should be full):";
00175 testRecord2.display();
00176
00177 //test constructor 3
00178 Grid grid_test(longitude_float, latitude_float);
00179
00180 Record testRecord3(zip, place, state, county, grid_test);
00181
00182 cout << "Constructor3 record (record should be full):";
00183 testRecord3.display();
00184
00185 //test display field
00186 cout << endl << "Test Display Field, display city:";
00187 testRecord3.display("CITY");
00188 cout << " expected: Little Falls" << endl;
00189
00190 cout << "Test Display Field, display state:";
00191 testRecord3.display("STATE");
00192 cout << " expected: Minnesota" << endl;
00193 }

```

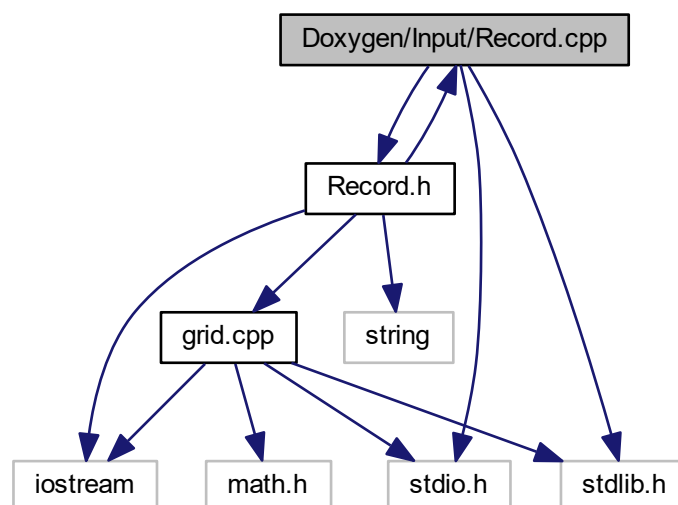
4.11 Doxygen/Input/Record.cpp File Reference

```

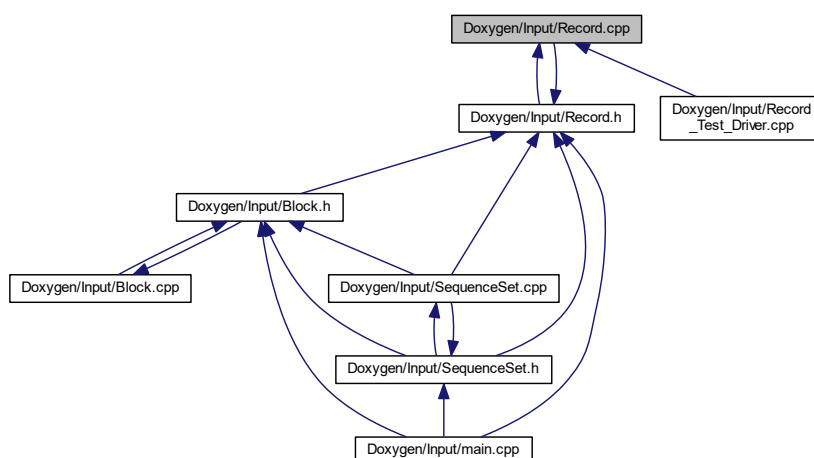
#include "Record.h"
#include <stdio.h>
#include <stdlib.h>

```

Include dependency graph for Record.cpp:



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



4.12 Record.cpp

```

00001
00017 #include "Record.h"
00018 #include <stdio.h>
00019 #include <stdlib.h>
00020 using namespace std;
00021
00022 Record::Record()
00023 {
  
```

```

00024     zip_code = "";
00025     place_name = "";
00026     state = "";
00027     county = "";
00028     this -> set_longitude_latitude( 0.0, 0.0 );
00029     if(DEBUG) {cout << "Made an empty record.\n";}
00030 }
00031
00032 Record::Record(string _zip_code, string _place_name, string _state, string _county, Grid _gridPoint)
00033 {
00034     zip_code = _zip_code;
00035     place_name = _place_name;
00036     state = _state;
00037     county = _county;
00038     this -> set_grid_point( _gridPoint );
00039     if(DEBUG) {cout << "Made a filled record using a gridPoint.\n";}
00040 }
00041
00042 Record::Record(string _zip_code, string _place_name, string _state, string _county, string latitude,
00043               string longitude)
00044 {
00045     float lon;
00046     float lat;
00047     try{
00048         lon = string_to_float( longitude );
00049     }
00050     catch(...){
00051         cout << "ERROR SETTING LONGITUDE, SETTING IT TO 0\n";
00052         lon = 0;
00053     }
00054     try{
00055         lat = string_to_float( latitude );
00056     }
00057     catch(...){
00058         cout << "ERROR SETTING LATITUDE IN " << zip_code << ", SETTING IT TO 0\n";
00059         lat = 0;
00060     }
00061 }
00062 zip_code = _zip_code;
00063 place_name = _place_name;
00064 state = _state;
00065 county = _county;
00066 this -> set_longitude_latitude( lon, lat );
00067 if(DEBUG) {cout << "Made a filled record using string lat/longs.\n";}
00068 }
00069
00070 void Record::display()
00071 {
00072     if(DEBUG) {cout << "Displaying the whole record from the record.\n";}
00073     cout << endl
00074           << "Zipcode:\t" << get_field("Zip")
00075           << "\nPlace:\t\t" << get_field("City")
00076           << "\nState:\t\t" << get_field("State")
00077           << "\nCounty:\t\t" << get_field("County")
00078           << "\nLongitude:\t" << get_field("Longitude")
00079           << "\nLatitude:\t" << get_field("Latitude")
00080           << endl;
00081 }
00082
00083 void Record::display(string field)
00084 {
00085     if(DEBUG) {cout << "Displaying the " << field << " portion of the record.\t";}
00086     for(int i = 0; field[i] != NULL; i++){
00087         field[i] = toupper(field[i]);
00088     }
00089     if(field=="Z" || field=="ZIP")
00090         cout << zip_code << endl;
00091     else if(field=="CITY" || field=="P" || field=="PLACE_NAME")
00092         cout << place_name << endl;
00093     else if(field=="STATE")
00094         cout << state << endl;
00095     else if(field=="COUNTY")
00096         cout << county << endl;
00097     else if(field=="G" || field=="GRID")
00098         cout << gridPoint.getLatitude() << " " << gridPoint.getLongitude() << endl;
00099     else if(field == "LAT" || field == "LATITUDE")
00100         cout << gridPoint.getLatitude() << endl;
00101     else if(field == "LONG" || field == "LONGITUDE")
00102         cout << gridPoint.getLongitude() << endl;
00103     else
00104         cout << "Invalid field has been entered." << endl;
00105 }
00106
00107 string Record::get_field(string field)
00108
00109

```

```

00110 {
00111     if(DEBUG) {cout << "Retrieving the " << field << " portion of the record.\t";}
00112     string returnString;
00113     for(int i = 0; field[i] != NULL; i++){
00114         field[i] = toupper(field[i]);
00115     }
00116
00117     if(field=="Z" || field=="ZIP")
00118         returnString = zip_code;
00119     else if(field=="CITY" || field=="P" || field=="PLACE_NAME")
00120         returnString = place_name;
00121     else if(field=="STATE")
00122         returnString = state;
00123     else if(field=="COUNTY")
00124         returnString = county;
00125     else if(field=="G" || field=="GRID")
00126         returnString = to_string(gridPoint.getLatitude()) + " " + to_string(gridPoint.getLongitude());
00127     else if(field == "LAT" || field == "LATITUDE")
00128         returnString = to_string(gridPoint.getLatitude());
00129     else if(field == "LONG" || field == "LONGITUDE")
00130         returnString = to_string(gridPoint.getLongitude());
00131     else
00132         returnString = "ERROR";
00133
00134     return returnString;
00135 }
00136
00137 void Record::set_field(string field, string data)
00138 {
00139     if(DEBUG) {cout << "Setting the " << field << " portion of the record from " << get_field(field) << " to" <<
data << ".\n";}
00140     for(int i = 0; field[i] != NULL; i++){
00141         field[i] = toupper(field[i]);
00142     }
00143
00144     for(int i = 0; data[i] != NULL; i++){
00145         data[i] = toupper(data[i]);
00146     }
00147
00148     if(field=="Z" || field=="ZIP")
00149         zip_code = data;
00150     else if(field=="CITY" || field=="P" || field=="PLACE_NAME")
00151         place_name = data;
00152     else if(field=="STATE")
00153         state = data;
00154     else if(field=="COUNTY")
00155         county = data;
00156     else if(field=="G" || field=="GRID")
00157         cout << "grid setter needs to implemented";
00158     else if(field == "LAT" || field == "LATITUDE")
00159         gridPoint.setLatitude(data);
00160     else if(field == "LONG" || field == "LONGITUDE")
00161         gridPoint.setLongitude(data);
00162     else
00163         cout << "ERROR" << endl;
00164 }
00165
00166 void Record::set_longitude_latitude(float longitude, float latitude)
00167 {
00168     gridPoint.setLatitude( latitude );
00169     gridPoint.setLongitude( longitude );
00170 }
00171
00172 void Record::set_grid_point(Grid _gridPoint)
00173 {
00174     gridPoint.setLatitude( _gridPoint.getLatitude() );
00175     gridPoint.setLongitude ( _gridPoint.getLongitude() );
00176 }
00177
00178 //helper functions
00179
00180 float Record::string_to_float(string str)
00181 {
00182     size_t size;
00183     float float_value = stof(str, &size);
00184
00185     return float_value;
00186 }
00187
00188
00189
00190
00191
00192
00193
00194
00195

```

```

00196
00197 // /**-----
00198 // * @Record.cpp
00199 // * Class Record (Contains information about individual zipcodes)
00200 // * @author Tyler Lahr, Ryan Sweeney, and Seth Pomahatch
00201 // * (Additional comments by Mark Christenson)
00202 // *-----
00203 // * Record class: Used by Sequence Set Class
00204 // * includes additional features:
00205 // * -- Display the whole record it represents
00206 // * -- Display a field with in the record
00207 // * -- Return a field as a string
00208 // * -- Return the latitude
00209 // * -- Return the longitude
00210 // *-----
00211 // */
00212
00213 // #include "Record.h"
00214 // #include <stdio.h>
00215 // #include <stdlib.h>
00216 // using namespace std;
00217
00218 // Record::Record()
00219 // {
00220 //     zip_code = "";
00221 //     place_name = "";
00222 //     state = "";
00223 //     county = "";
00224 //     this -> set_longitude_latitude( 0.0, 0.0 );
00225 //     if(DEBUG) {cout << "Made an empty record.\n";}
00226 // }
00227
00228 // Record::Record(string _zip_code, string _place_name, string _state, string _county, Grid
    _gridPoint)
00229 // {
00230 //     zip_code = _zip_code;
00231 //     place_name = _place_name;
00232 //     state = _state;
00233 //     county = _county;
00234 //     this -> set_grid_point( _gridPoint );
00235 //     if(DEBUG) {cout << "Made a filled record using a gridPoint.\n";}
00236 // }
00237
00238 // Record::Record(string _zip_code, string _place_name, string _state, string _county, string
    latitude, string longitude)
00239 // {
00240 //     float lon = string_to_float( longitude );
00241 //     float lat = string_to_float( latitude );
00242 //
00243 //     zip_code = _zip_code;
00244 //     place_name = _place_name;
00245 //     state = _state;
00246 //     county = _county;
00247 //     this -> set_longitude_latitude( lon, lat );
00248 //     if(DEBUG) {cout << "Made a filled record using string lat/longs.\n";}
00249 // }
00250
00251 // void Record::display()
00252 // {
00253 //     if(DEBUG) {cout << "Displaying the whole record from the record.\n";}
00254 //     cout << endl
00255 //         << "Zipcode: " << get_field("Zip")
00256 //         << " Place: " << get_field("Place_name")
00257 //         << " State: " << get_field("State")
00258 //         << " County: " << get_field("County")
00259 //         << " Longitude: " << get_field("Longitude")
00260 //         << " Latitude: " << get_field("Latitude")
00261 //         << endl;
00262 // }
00263
00264 // void Record::display(string field)
00265 // {
00266 //     if(DEBUG) {cout << "Displaying the " << field << " portion of the record.\n";}
00267 //     for(int i = 0; field[i] != NULL; i++){
00268 //         field[i] = toupper(field[i]);
00269 //     }
00270 //
00271 //     if(field=="Z" || field=="ZIP")
00272 //         cout << zip_code << endl;
00273 //     else if(field=="CITY" || field=="P" || field=="PLACE_NAME")
00274 //         cout << place_name << endl;
00275 //     else if(field=="STATE")
00276 //         cout << state << endl;
00277 //     else if(field=="COUNTY")
00278 //         cout << county << endl;
00279 //     else if(field=="G" || field=="GRID")
00280 //         cout << gridPoint.getLatitude() << " " << gridPoint.getLongitude() << endl;

```

```

00281 //     else if(field == "LAT" || field == "LATITUDE")
00282 //         cout << gridPoint.getLatitude() << endl;
00283 //     else if(field == "LONG" || field == "LONGITUDE")
00284 //         cout << gridPoint.getLongitude() << endl;
00285 //     else
00286 //         cout << "Invalid field has been entered." << endl;
00287 // }
00288
00289 // string Record::get_field(string field)
00290 // {
00291 //     if(DEBUG) {cout << "Retrieving the " << field << " portion of the record.\n";}
00292 //     string returnString;
00293 //     for(int i = 0; field[i] != NULL; i++){
00294 //         field[i] = toupper(field[i]);
00295 //     }
00296
00297 //     if(field=="Z" || field=="ZIP")
00298 //         returnString = zip_code;
00299 //     else if(field=="CITY" || field=="P" || field=="PLACE_NAME")
00300 //         returnString = place_name;
00301 //     else if(field=="STATE")
00302 //         returnString = state;
00303 //     else if(field=="COUNTY")
00304 //         returnString = county;
00305 //     else if(field=="G" || field=="GRID")
00306 //         returnString = to_string(gridPoint.getLatitude()) + " " + to_string(gridPoint.getLongitude());
00307 //     else if(field == "LAT" || field == "LATITUDE")
00308 //         returnString = to_string(gridPoint.getLatitude());
00309 //     else if(field == "LONG" || field == "LONGITUDE")
00310 //         returnString = to_string(gridPoint.getLongitude());
00311 //     else
00312 //         returnString = "ERROR";
00313
00314 //     return returnString;
00315 // }
00316
00317 // void Record::set_field(string field, string data)
00318 // {
00319 //     if(DEBUG) {cout << "Setting the " << field << " portion of the record from " << get_field(field) << "
00320 //         to" << data << ".\n";}
00321 //     for(int i = 0; field[i] != NULL; i++){
00322 //         field[i] = toupper(field[i]);
00323 //     }
00324 //     for(int i = 0; data[i] != NULL; i++){
00325 //         data[i] = toupper(data[i]);
00326 //     }
00327
00328 //     if(field=="Z" || field=="ZIP")
00329 //         zip_code = data;
00330 //     else if(field=="CITY" || field=="P" || field=="PLACE_NAME")
00331 //         place_name = data;
00332 //     else if(field=="STATE")
00333 //         state = data;
00334 //     else if(field=="COUNTY")
00335 //         county = data;
00336 //     else if(field=="G" || field=="GRID")
00337 //         cout << "grid setter needs to implemented";
00338 //     else if(field == "LAT" || field == "LATITUDE")
00339 //         gridPoint.setLatitude(data);
00340 //     else if(field == "LONG" || field == "LONGITUDE")
00341 //         gridPoint.setLongitude(data);
00342 //     else
00343 //         cout << "ERROR" << endl;
00344 // }
00345
00346 // void Record::set_longitude_latitude(float longitude, float latitude)
00347 // {
00348 //     gridPoint.setLatitude( latitude );
00349 //     gridPoint.setLongitude( longitude );
00350 // }
00351
00352 // void Record::set_grid_point(Grid _gridPoint)
00353 // {
00354 //     gridPoint.setLatitude( _gridPoint.getLatitude() );
00355 //     gridPoint.setLongitude ( _gridPoint.getLongitude() );
00356 // }
00357
00358 // float Record::string_to_float(string str)
00359 // {
00360 //     size_t size;
00361 //     float float_value = stof(str, &size);
00362
00363 //     return float_value;
00364 // }
00365
00366 // //     if gridPoint.setLatitude(data);

```

```

00367 // //   else if(field == "LONG" || field == "LONGITUDE")
00368 // //       gridPoint.setLongitude(data);
00369 // //   else
00370 // //       cout << "ERROR" << endl;
00371 // // }
00372
00373 // void Record::set_longitude_latitude(float longitude, float latitude)
00374 // {
00375 //     gridPoint.setLatitude( latitude );
00376 //     gridPoint.setLongitude( longitude );
00377 // }
00378
00379 // void Record::set_grid_point(Grid _gridPoint)
00380 // {
00381 //     gridPoint.setLatitude( _gridPoint.getLatitude() );
00382 //     gridPoint.setLongitude ( _gridPoint.getLongitude() );
00383 // }
00384
00385 // float Record::string_to_float(string str)
00386 // {
00387 //     size_t size;
00388 //     float float_value = stof(str, &size);
00389 //
00390 //     return float_value;
00391 // }

```

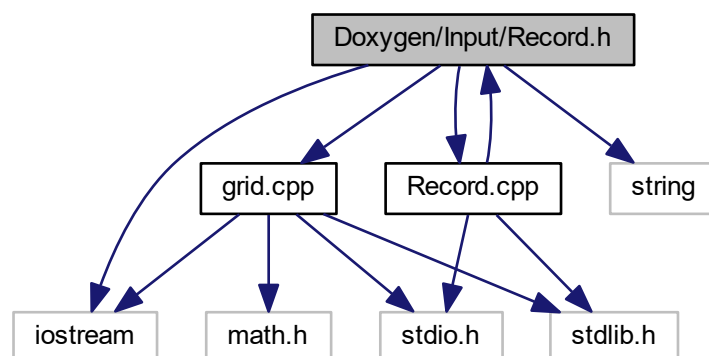
4.13 Doxygen/Input/Record.h File Reference

```

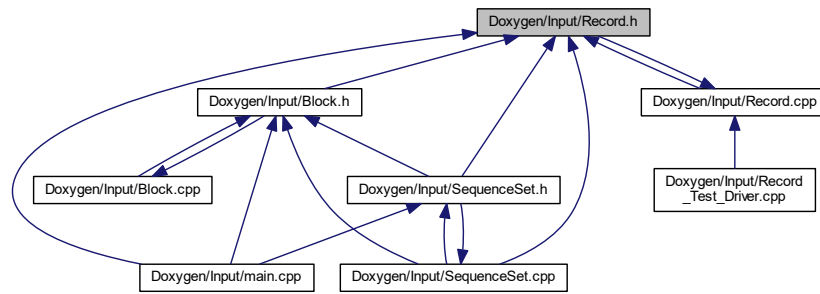
#include <iostream>
#include <string>
#include "grid.cpp"
#include "Record.cpp"

```

Include dependency graph for Record.h:



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



Classes

- class [Record](#)

4.14 Record.h

```

00001
00017 #ifndef RECORD_H
00018 #define RECORD_H
00019
00020 #include <iostream>
00021 #include <string>
00022 #include "grid.cpp"
00023 using namespace std;
00024
00025 class Record
00026 {
00027 public:
00032     Record();
00037     Record(string, string, string, string, Grid);
00042     Record(string, string, string, string, string, string);
00047     void display();
00052     void display(string); //This might benefit from calling get_field
00057     string get_field(string); //This should have a switch statement
00063     void set_field(string, string);
00068     void set_longitude_latitude(float, float);
00073     void set_grid_point(Grid);
00074
00075 private:
00076     bool isEmpty;
00077     string zip_code;
00078     string place_name;
00079     string state;
00080     string county;
00081     Grid gridPoint;
00086     float string_to_float(string);
00087 };
00088
00089 #include "Record.cpp"
00090
00091 #endif
  
```

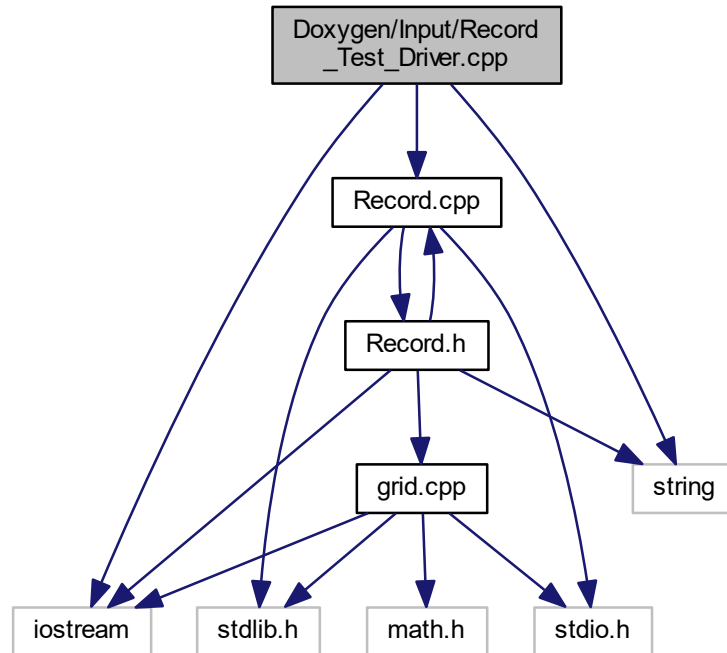
4.15 Doxygen/Input/Record_Test_Driver.cpp File Reference

```

#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](#).

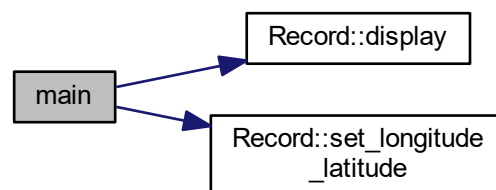
4.15.2 Function Documentation

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

```

00001 //g++ -std=c++11 -o record_test Record_Test_Driver
00002
00003 #include "Record.cpp"
00004 #include<iostream>
00005 #include<string>
00006
00007 using namespace std;
00008
00009 enum Field
00010 {
00011     Z,
00012     ZIP,
00013     CITY,

```

```

00014     P,
00015     PLACE_NAME,
00016     STATE,
00017     COUNTY,
00018     G,
00019     GRID
00020 };
00021
00022 int main()
00023 {
00024     //test default constructor
00025     Record testRecord;
00026     cout << "Default constructor record:";
00027     testRecord.display();
00028
00029     //test fill record
00030     string zip = "56345";
00031     string place = "Little Falls";
00032     string state = "Minnesota";
00033     string county = "Morrison";
00034     float longitude = 74.25;
00035     float latitude = 79.72;
00036
00037     testRecord.set_zip_code( zip );
00038     testRecord.set_place_name( place );
00039     testRecord.set_state( state );
00040     testRecord.set_county( county );
00041     testRecord.set_longitude_latitude( longitude, latitude );
00042
00043     cout << "Filled Record:";
00044     testRecord.display();
00045
00046     //test constructor 2
00047     string longitude_string = "74.25";
00048     string latitude_string = "79.72";
00049
00050     Record testRecord2(zip, place, state, county, longitude_string, latitude_string);
00051
00052     cout << "Constructor2 record (long/lat are strings):";
00053     testRecord2.display();
00054
00055     //test constructor 3
00056     Grid grid_test(longitude, latitude);
00057
00058     Record testRecord3(zip, place, state, county, grid_test);
00059
00060     cout << "Constructor3 record (long/lat are gridPoint):";
00061     testRecord3.display();
00062
00063     cout << endl << "check enum:";
00064     testRecord3.display(CITY);
00065     cout << " expected: Little Falls" << endl;
00066
00067     testRecord3.display(STATE);
00068     cout << " expected: Minnesota" << endl;
00069
00070     return 0;
00071 }

```

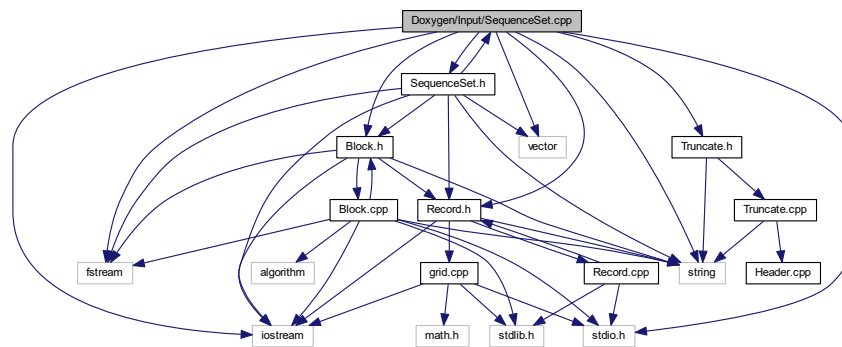
4.17 Doxygen/Input/SequenceSet.cpp File Reference

```

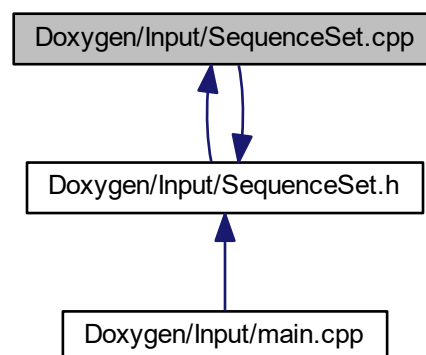
#include "SequenceSet.h"
#include <iostream>
#include "Truncate.h"
#include "Record.h"
#include "Block.h"
#include <string>
#include <fstream>
#include <vector>
#include <stdio.h>

```

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()

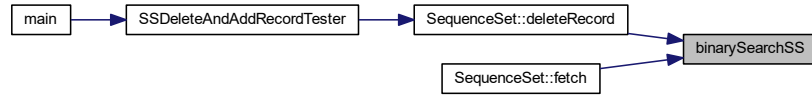
```
int binarySearchSS (
    const string arr[],
```

```

    string x,
    int n )

```

Here is the caller graph for this function:



4.18 SequenceSet.cpp

```

00001
00014 #include "SequenceSet.h"
00015 #include <iostream>
00016 #include "Truncate.h"
00017 #include "Record.h"
00018 #include "Block.h"
00019 #include "SequenceSet.h"
00020 #include <string>
00021 #include <fstream>
00022 #include <vector>
00023 #include <stdio.h>
00024
00025 using namespace std;
00026
00027 //binarySearch recycled from block
00028 int binarySearchSS(const string arr[], string x, int n);
00029
00030 SequenceSet::SequenceSet(){
00031     ofstream SSFile;
00032     SSFile.open(SSFileName);
00033     SSFile << "Sequence Set File\n";
00034     SSFile.close();
00035     recordCount = getRecordCount();
00036     fillIndex();
00037     Block * currentBlock = headBlock;
00038     blockCount = 0;
00039     for(unsigned long long i = 0; i < recordCount; i++){
00040         if(i%BLOCKFILLCOUNT == 0 && i != 0){
00041             if(DEBUG){cout << "Making a new block for the chain." << endl;}
00042             blockCount++;
00043             Block * newBlock = new Block(blockCount);
00044             currentBlock->setNextBlock(newBlock);
00045             newBlock->setPrevBlock(currentBlock);
00046             currentBlock = newBlock;
00047         }
00048         if(DEBUG){cout<<"Passing "<<to_string(pKeyIndex.at(i))<<" into the add function."<<endl;}
00049         currentBlock->addRecord(to_string(pKeyIndex.at(i)));
00050     }
00051     writeBlocks();
00052
00053     //reset the record avail list
00054     ofstream recordAvailList;
00055     recordAvailList.open(recordAvailListFileName);
00056     recordAvailList << "";
00057     recordAvailList.close();
00058 }
00059
00060 unsigned long long SequenceSet::headerLength(string _fileName){
00061     fstream data;
00062     unsigned long long length = 0;
00063     unsigned long long L = 0;
00064     data.open(_fileName);
00065     string str;
00066
00067     if(DEBUG){cout << "String outside while loop, in headerLength: " << str << endl;}
00068     while(data.peek() != EOF){
00069         if(DEBUG && false){cout << "String in headerLength: " << str << endl;}
00070         getline(data, str);
00071         length += str.length();
00072         length++;
00073         if(str == HEADERENDSTRING){
00074             L = length;

```

```

00075         if(DEBUG){cout<<"L defined: " << L << "\n";}
00076     }
00077 }
00078
00079 data.close();
00080
00081 return L;
00082 }
00083
00084 unsigned int SequenceSet::getRecordCount(){
00085     string fileName = DATAFILENAME;
00086     string field;
00087     string str = "";
00088     char c;
00089     fstream data;
00090     unsigned int recordCount = 0;
00091     data.open(fileName);
00092     getline(data, field); //Skip title
00093     while(data.peek() != ':'){
00094         data.get(c);
00095         field += c;
00096         if(DEBUG && true){cout << "Char c: " << c << endl;}
00097     }
00098
00099     //This while is to skip non number values before approaching what to do with the values
00100     while(data.peek() < '0' || data.peek() > '9'){
00101         data.get(c);
00102     }
00103     getline(data, str);
00104
00105     recordCount = stoi(str);
00106
00107     if(DEBUG){cout << "String: " << str << "\nrecords: " << recordCount << endl;}
00108     if(field == "Records"){
00109         getline(data, str);
00110         recordCount = stoi(str);
00111         if(DEBUG){cout << "Record Count: " << recordCount << endl;}
00112     }
00113     data.close();
00114
00115     return recordCount;
00116 }
00117
00118 void SequenceSet::fillIndex(){
00119     string field;
00120     string str = "";
00121     char c;
00122     fstream data;
00123
00124     data.open("RecordOffsets.txt");
00125
00126     for(unsigned int i = 0; i < recordCount; i++){
00127         string recordData = "";
00128         getline(data, recordData);
00129         if(DEBUG){cout << "recordData: " << recordData << endl;}
00130         str = "";
00131         for(int j = 0; j < ZIPLLENGTH; j++){
00132             str += recordData[j];
00133         }
00134         //index[i][0] = stoi(str); //five chars of string
00135         pKeyIndex.push_back(stoi(str));
00136         if(DEBUG){cout << "String: " << str << endl;}
00137         if(DEBUG){cout << "pKeyIndex.at(i): " << pKeyIndex.at(i) << endl;}
00138         str = "";
00139         for(int j = ZIPLLENGTH; j < recordData.length(); j++){
00140             str += recordData[j];
00141         }
00142         //index[i][1] = stoi(str); //the rest of the string
00143         offsetIndex.push_back(stoi(str));
00144         if(DEBUG){cout << "String: " << str << endl;}
00145         if(DEBUG){cout << "offsetIndex.at(" << i << "): " << offsetIndex.at(i) << endl;}
00146     }
00147     data.close();
00148 }
00149
00150 string SequenceSet::fetch(string pKey){
00151     fstream data;
00152     data.open(DATAFILENAME);
00153     string returnString = "";
00154     for(int i = ZIPLLENGTH - pKey.length(); i > 0; i--){
00155         if(DEBUG){cout << "For loop in fetch. i = " << i << endl;}
00156         returnString += " ";
00157     }
00158     returnString = pKey;
00159     returnString += " not found.\n";
00160
00161     int position;

```

```

00162     if(pKey != ""){
00163         position = binarySearchSS(pKey);
00164     }
00165     if(DEBUG) {cout << "Searching "< pKey << " returned: " << position << endl;}
00166     if(position>=0 && pKey != ""){
00167         data.seekg(offsetIndex.at(binarySearchSS(pKey)));
00168         getline(data, returnString);
00169     }
00170     data.close();
00171
00172     return returnString;
00173 }
00174
00175 string SequenceSet::fetch(unsigned int pKey){
00176     return fetch(to_string(pKey));
00177 }
00178
00179 void SequenceSet::makeRecordOffsets(string fileName){
00180     string zip = " ";
00181     fstream data;
00182     ofstream index;
00183     string str;
00184     index.open("RecordOffsets.txt");
00185     unsigned long long offset = headerLength(fileName);
00186     data.open(fileName);
00187     data.seekg(offset);
00188
00189     if(DEBUG && false){cout << "String in makeRecordOffsets is: " << str << endl;}
00190     getline(data, str);
00191
00192     while(data.peek() != EOF){
00193         if (DEBUG && false){cout << str << endl;}
00194         for(int i = 0; i < ZIPLength; i++){
00195             zip[i] = str[i];
00196         }
00197         if(DEBUG && false){cout<<zip<< " is at " << offset << endl;}
00198         index << zip << offset << endl;
00199         getline(data, str);
00200         offset += str.length();
00201         offset++;
00202     }
00203
00204     data.close();
00205     index.close();
00206 }
00207
00212 int SequenceSet::binarySearchSS(string x)
00213 {
00214     //int int_arr[n];
00215     unsigned int n = recordCount;
00216     int int_string;
00217     /*
00218     //convert the records (array of strings) to array of int
00219     for (unsigned int i = 0; i < n; i++)
00220     {
00221         if(arr[i] != null_str)
00222             int_arr[i] = stoi(arr[i]);
00223     }
00224     */
00225     //convert string to find to int
00226     if(DEBUG){cout << "(stoi)ing this string: \"" << x << "\"\n";}
00227     try{
00228         int_string = stoi(x);    unsigned int l = 0 ;
00229         unsigned int r = n - 1;
00230         while (l <= r)
00231         {
00232             int m = l + (r - l) / 2;
00233             if(DEBUG) {cout << "mid: " << m << endl;}
00234
00235             //if(DEBUG) {cout << "comparing " << int_string << " and " << int_arr[m] << endl;}
00236             if(DEBUG) {cout << "comparing " << int_string << " and " << pKeyIndex.at(m) << endl;}
00237
00238             if ( pKeyIndex.at(m) == int_string ){
00239                 if(DEBUG) {cout << "record found" << endl;}
00240                 return m;
00241             }
00242
00243             // If x is greater, ignore left half
00244             if ( pKeyIndex.at(m) < int_string ){
00245                 l = m + 1;
00246                 if(DEBUG) {cout << "new l: " << l << endl;}
00247             }
00248
00249             // If x is smaller, ignore right half
00250             else{
00251                 r = m - 1;
00252                 if(DEBUG) {cout << "new r: " << l << endl;}

```



```

00253     }
00254     }
00255 }
00256     catch(...){cout << "ERROR (stoi)ING THIS STRING: \"\" « x « "\"\n";}
00257
00258     return -1;
00259 }
00260
00261 Record SequenceSet::fillRecord(string RecordString){
00262     string zip_code, place_name, state, county, latitude, longitude;
00263     int position = 0;
00264     if(DEBUG){cout << "In fillRecord for Sequence Set Class\n\tRecordString: "
00265         << RecordString << endl;}
00266     zip_code = "";
00267     for(auto i = 0; i < ZIPLENGTH ; i++){
00268         if(RecordString[position] != ' '){
00269             zip_code += RecordString[position];
00270         }
00271         position++;
00272     }
00273
00274     place_name = "";
00275     for(int i = 0; i < 31/*Length of place name*/; i++){
00276         if(RecordString[position] != ' '){
00277             place_name += RecordString[position];
00278         }
00279         position++;
00280     }
00281
00282     state = "";
00283     for(int i = 0; i < 2/*Length of state*/; i++){
00284         if(RecordString[position] != ' '){
00285             state += RecordString[position];
00286         }
00287         position++;
00288     }
00289
00290     county = "";
00291     for(int i = 0; i < 38/*Length of county*/; i++){
00292         if(RecordString[position] != ' '){
00293             county += RecordString[position];
00294         }
00295         position++;
00296     }
00297
00298     latitude = "";
00299     for(int i = 0; i < 9/*Length of latitude*/; i++){
00300         if(RecordString[position] != ' '){
00301             latitude += RecordString[position];
00302         }
00303         position++;
00304     }
00305
00306     longitude = "";
00307     for(int i = 0; i < 8/*Length of longitude*/; i++){
00308         if(RecordString[position] != ' '){
00309             longitude += RecordString[position];
00310         }
00311         position++;
00312     }
00313     if(DEBUG){cout << "\tRecordElements: " << "\n\t\t"
00314         << zip_code << "\n\t\t" << place_name << "\n\t\t"
00315         << state << "\n\t\t" << county << "\n\t\t"
00316         << latitude << "\n\t\t" << longitude << endl;}
00317
00318     Record returnRecord(zip_code, place_name, state, county, latitude, longitude);
00319
00320     if(DEBUG){returnRecord.display();}
00321
00322     return returnRecord;
00323 }
00324
00325 void SequenceSet::writeBlocks(){
00326     Block * currentBlock = headBlock;
00327     for(auto i = 0; i < blockCount; i++){
00328         if(DEBUG){cout << "Writing block " << i << " from the chain." << endl;}
00329         currentBlock->write(SSFileName);
00330         currentBlock = currentBlock->getNextBlock();
00331     }
00332 }
00333
00334 void SequenceSet::fillRecordBlock(unsigned long long blockID){
00335     string str, zip, passed;
00336     Block * currentBlock = headBlock;
00337     for(auto i = 0; i < blockID; i++){
00338         currentBlock = currentBlock->getNextBlock();
00339     }

```

```

00340
00341     currentBlock->getRecords(recordBlock);
00342     for(auto i = 0; i < currentBlock->getRecordCount(); i++){
00343         passed = fetch(recordBlock[i].get_field("ZIP"));
00344         if(DEBUG){
00345             cout << "\n*****"
00346                  << "\nString passed to fill record: " << passed << endl;
00347         }
00348         if(passed != " not found.\n" && passed != " not found."){
00349             recordBlock[i] = fillRecord(passed);
00350             if(DEBUG){recordBlock[i].display();}
00351         }
00352     }
00353 }
00354
00355 void SequenceSet::addBlockStateKey(unsigned long long blockID){
00356     fillRecordBlock(blockID);
00357     Block * currentBlock = headBlock;
00358
00359     for(auto i = 0; i < blockID; i++){
00360         currentBlock = currentBlock->getNextBlock();
00361     }
00362
00363     for(auto i = 0; i < currentBlock->getRecordCount(); i++){
00364         string state = recordBlock[i].get_field("state");
00365
00366         for(auto i = 0; i < RECORDSPERBLOCK; i++){
00367             string state = recordBlock[i].get_field("state");
00368
00369             if(state != ""){
00370                 bool stateFound = false;
00371                 unsigned int index = 0;
00372
00373                 if(stateZips.size() == 0){
00374                     vector <string> newRow;
00375                     newRow.push_back(state);
00376                     stateZips.push_back(newRow);
00377                 }
00378
00379                 while(index < stateZips.size() && !stateFound){
00380                     if(stateZips[index].at(0) == state){
00381                         if(DEBUG){cout << "Found " << state << " at index = " << index << endl;}
00382                         stateFound = true;
00383                     }
00384                     else{index++;}
00385                 }
00386
00387                 if(!stateFound){
00388                     if(DEBUG){cout << state<<" not found.\n";}
00389                     vector <string> newRow;
00390                     newRow.push_back(state);
00391                     stateZips.push_back(newRow);
00392                     if(DEBUG){cout << stateZips[index].at(0)<<" pushed successfully.\n";}
00393                     if(DEBUG){
00394                         stateZips[index].push_back(":");
00395                         cout << "Pushing a smily :) \n";
00396                         cout << stateZips[index].at(1) << endl;
00397                         stateZips[index].pop_back();
00398                     }
00399                 }
00400
00401                 if(DEBUG){cout << "Pushing " << recordBlock[i].get_field("zip") << " to " << index << " column.\n";}
00402                 stateZips[index].push_back(recordBlock[i].get_field("zip"));
00403                 //if(DEBUG){cout << stateZips[index].at(stateZips[index].size())<<" pushed successfully.\n";}
00404
00405
00406                 cout << stateZips[index].at(0) << ": "
00407                      << stateZips[index].at(stateZips[index].size()-1) << endl;
00408             }
00409         }
00410     }
00411 }
00412
00413
00414 bool SequenceSet::deleteRecord(int pKey)
00415 {
00416     //search if the record is in the sequence set
00417     int position = binarySearchSS( to_string(pKey) );
00418     if(DEBUG){cout << "Searching for " << pKey << " returned: " << position << endl;}
00419     if(position == -1){
00420         cout << "Record does not exist in Sequence Set." << endl;
00421         return false;
00422     }
00423     else{
00424         //add deleted record offset to avail list
00425         string strTemp = "";
00426         string newString = "";

```

```

00427     fstream recordAvailListIn;
00428     recordAvailListIn.open(recordAvailListFileName);
00429     while(recordAvailListIn.peek() != EOF){
00430         strTemp += recordAvailListIn.get();
00431         cout << strTemp << endl;
00432     }
00433     newString = to_string( offsetIndex.at(position) ) + "/" + to_string( position ) + "\n" + strTemp;
00434     cout << newString << " result" << endl;
00435     recordAvailListIn.close();
00436
00437     ofstream recordAvailList;
00438     recordAvailList.open(recordAvailListFileName);
00439     recordAvailList << newString;
00440     recordAvailList.close();
00441
00442     //delete record from us_postal_codes.txt
00443     fstream usPostalCodes;
00444     usPostalCodes.open("us_postal_codes.txt");
00445     usPostalCodes.seekg(offsetIndex.at(position));
00446     for(int i = 0; i < 94; i++){ //94 is the length of record
00447         usPostalCodes << " ";
00448     }
00449     usPostalCodes.close();
00450
00451     //delete record in index vector
00452     pKeyIndex.erase(pKeyIndex.begin() + position);
00453     offsetIndex.erase(offsetIndex.begin() + position);
00454     if(DEBUG) {position = binarySearchSS( to_string(pKey) );}
00455     if(DEBUG) {cout << "Deleted record in index vector. Researching for "<< pKey << " returned: " <<
position << endl;}
00456     recordCount--; //decrement the total record count
00457
00458     //delete record in linked list of blocks
00459     Block * currentBlock = headBlock;
00460     for(auto i = 0; i < blockCount; i++){
00461         if(DEBUG){cout << "Searching block "<< i << " from the chain." << endl;}
00462         if( pKey <= currentBlock->getLastRecordPKey() ){
00463             currentBlock->deleteRecord( to_string(pKey) );
00464             break;
00465         }
00466         else{
00467             currentBlock = currentBlock->getNextBlock();
00468         }
00469     }
00470
00471     //merge blocks if needed
00472     if( currentBlock->getRecordCount() < RECORDSPERBLOCK / 2 ){
00473         //check next block to see if it can merge
00474         if( (currentBlock->getNextBlock())->getRecordCount() == RECORDSPERBLOCK / 2 ){
00475             currentBlock->getRecords( recordBlock ); //get the pkeys
00476             for(int i = 0; i < currentBlock->getRecordCount(); i++){
00477                 (currentBlock->getNextBlock())->addRecord(recordBlock[i].get_field("zip"));
00478                 currentBlock->deleteRecord( recordBlock[i].get_field("zip") );
00479             }
00480             //add the pointer to the current block to the avail vector
00481             blockAvailList.push_back( currentBlock );
00482             //change the pointers to avoid the empty block
00483             currentBlock->getPreviousBlock()->setNextBlock( currentBlock->getNextBlock() );
00484             currentBlock->getNextBlock()->setPrevBlock( currentBlock->getPreviousBlock() );
00485             blockCount--;
00486         }
00487         //check if previous block can merge
00488         else if( (currentBlock->getPreviousBlock())->getRecordCount() == RECORDSPERBLOCK / 2 ){
00489             currentBlock->getRecords( recordBlock ); //get the pkeys
00490             for(int i = 0; i < currentBlock->getRecordCount(); i++){
00491                 (currentBlock->getPreviousBlock())->addRecord(recordBlock[i].get_field("zip"));
00492                 currentBlock->deleteRecord( recordBlock[i].get_field("zip") );
00493             }
00494             //add the pointer to the current block to the avail vector
00495             blockAvailList.push_back( currentBlock );
00496             //change the pointers to avoid the empty block
00497             currentBlock->getPreviousBlock()->setNextBlock( currentBlock->getNextBlock() );
00498             currentBlock->getNextBlock()->setPrevBlock( currentBlock->getPreviousBlock() );
00499             blockCount--;
00500         }
00501         //check if next block can redistribute
00502         else if( (currentBlock->getNextBlock())->getRecordCount() > RECORDSPERBLOCK / 2 ){
00503             (currentBlock->getNextBlock())->getRecords( recordBlock ); //get the pkeys
00504             currentBlock->addRecord( recordBlock[0].get_field("zip") );
00505             (currentBlock->getNextBlock())->deleteRecord( recordBlock[0].get_field("zip") );
00506         }
00507         //check if previous block can redistribute ???Will Never
Happen????????????????????????????????????????????????????????????????????????????????????????????
00508         // else if( (currentBlock->getPreviousBlock())->getRecordCount() > RECORDSPERBLOCK / 2 ){
00509         //     (currentBlock->getPreviousBlock())->getRecords( recordBlock ); //get the pkeys
00510         //     currentBlock->addRecord( recordBlock[0].get_field("zip") );
00511         //     (currentBlock->getNextBlock())->deleteRecord( recordBlock[0].get_field("zip") );

```

```

00512     //}
00513     }
00514
00515     return true;
00516 }
00517 }
00518
00519 string SequenceSet::extremeCoord(string state, char direction)
00520 {
00521     direction = toupper(direction);
00522     float extremePoint = 0;
00523     string zip = "";
00524     Record currentRecord;
00525     string str = state;
00526
00527     bool found = false;
00528     unsigned int index = 0;
00529     while(index < stateZips.size() && !found){
00530         if(stateZips[index][0] == str){found = true;}
00531         else{index++;}
00532     }
00533     currentRecord = fillRecord(fetch(stateZips[index][1]));
00534
00535     switch(direction)
00536     {
00537         case 'N':
00538         {
00539             extremePoint = stof(currentRecord.get_field("Lat"));
00540             zip = currentRecord.get_field("zip");
00541             for(int i = 1; i < stateZips[index].size(); i++)
00542             {
00543                 currentRecord = fillRecord(fetch(stateZips[index][i]));
00544                 if(extremePoint < stof(currentRecord.get_field("Lat")))
00545                 {
00546                     zip = currentRecord.get_field("zip");
00547                     extremePoint = stof(currentRecord.get_field("Lat"));
00548                 }
00549             }
00550         }
00551         break;
00552
00553         case 'E':
00554         {
00555             extremePoint = stof(currentRecord.get_field("Long"));
00556             zip = currentRecord.get_field("zip");
00557             for(int i = 1; i < stateZips[index].size(); i++)
00558             {
00559                 currentRecord = fillRecord(fetch(stateZips[index][i]));
00560                 if(extremePoint < stof(currentRecord.get_field("Long")))
00561                 {
00562                     zip = currentRecord.get_field("zip");
00563                     extremePoint = stof(currentRecord.get_field("Long"));
00564                 }
00565             }
00566         }
00567         break;
00568
00569         case 'S':
00570         {
00571             extremePoint = stof(currentRecord.get_field("Lat"));
00572             zip = currentRecord.get_field("zip");
00573             for(int i = 1; i < stateZips[index].size(); i++)
00574             {
00575                 currentRecord = fillRecord(fetch(stateZips[index][i]));
00576                 if(extremePoint > stof(currentRecord.get_field("Lat")))
00577                 {
00578                     zip = currentRecord.get_field("zip");
00579                     extremePoint = stof(currentRecord.get_field("Lat"));
00580                 }
00581             }
00582         }
00583         break;
00584
00585         case 'W':
00586         {
00587             extremePoint = stof(currentRecord.get_field("Long"));
00588             zip = currentRecord.get_field("zip");
00589             for(int i = 1; i < stateZips[index].size(); i++)
00590             {
00591                 currentRecord = fillRecord(fetch(stateZips[index][i]));
00592                 if(extremePoint > stof(currentRecord.get_field("Long")))
00593                 {
00594                     zip = currentRecord.get_field("zip");
00595                     extremePoint = stof(currentRecord.get_field("Long"));
00596                 }
00597             }
00598         }
00599     }

```

```

00599         break;
00600
00601     default:
00602     {
00603         cout << "UNDEFINED OPTION\n";
00604     }
00605     }
00606     return zip;
00607 }
00608
00609 int SequenceSet::test(){
00610     string field;
00611     string str = "";
00612     char c;
00613     fstream data;
00614
00615     int randomRecord = rand() % recordCount;
00616     //cout << "Retrieving record: " << index[randomRecord][0] << endl;
00617     cout << "Retrieving record: " << pKeyIndex.at(randomRecord) << endl;
00618     data.open(DATAFILENAME);
00619     //data.seekg(index[randomRecord][1]);
00620     data.seekg(offsetIndex.at(randomRecord));
00621     getline(data, str);
00622     cout << str << endl;
00623
00624     cout << fetch(1721) << endl;
00625     fillRecordBlock(88);
00626
00627     for(auto i = 0; i < RECORDSPERBLOCK; i++){
00628         if(DEBUG){cout << "\n*****\n";}
00629         recordBlock[i].display();
00630     }
00631
00632     sKeyStateBuilder();
00633
00634     unsigned int index = 0;
00635     unsigned int record = 1;
00636     Record currentRecord;
00637
00638     str = "MN";
00639     bool found = false;
00640     while(index < stateZips.size() && !found){
00641         if(stateZips[index][0] == str){found = true;}
00642         else{index++;}
00643     }
00644
00645     while(record < stateZips[index].size()){
00646         str = fetch(stateZips[index][record]);
00647         cout << str << endl;
00648         currentRecord = fillRecord(str);
00649         currentRecord.display();
00650         record++;
00651     }
00652
00653     cout << extremeCoord(str, 'n') << endl;
00654
00655     return 0;
00656 }
00657
00658
00659 void SequenceSet::sKeyStateBuilder(){
00660     if(DEBUG){cout << "Building sKeys for states.\n";}
00661     Block * currentBlock = headBlock;
00662     unsigned int index = 0;
00663     while(currentBlock!=NULL){
00664         addBlockStateKey(index);
00665         currentBlock = currentBlock->getNextBlock();
00666         index++;
00667     }
00668 }
00669
00670 void SequenceSet::addRecord(Record record)
00671 {
00672     //search record in linked list of blocks
00673     Block * currentBlock = headBlock;
00674     for(auto i = 0; i < blockCount; i++){
00675         if(DEBUG){cout << "Searching block "<< i << " from the chain." << endl;}
00676         if( stoi( record.get_field("zip") ) <= currentBlock->getLastRecordPKey() ){ //find the right
00677             block
00678             if(currentBlock->getRecordCount() == RECORDSPERBLOCK){ //if the block is full, do block
00679                 splitting
00680                 if( !blockAvailList.empty() ){ //if there exists a current empty block
00681                     Block* tempBlockPtr = blockAvailList.back(); //get the pointer to the empty block
00682                     blockAvailList.pop_back(); //delete the pointer from the avail list
00683                     //add the relative block to the linked list
00684                     tempBlockPtr->setNextBlock( currentBlock->getNextBlock() );
00685                     tempBlockPtr->setPrevBlock( (currentBlock->getNextBlock())->getPreviousBlock() );

```

```

00684         (currentBlock->getNextBlock())->setPrevBlock(tempBlockPtr);
00685         currentBlock->setNextBlock(tempBlockPtr);
00686         //split the data into the new block number
00687         currentBlock->getRecords( recordBlock ); //get the pkeys
00688         for(int i = RECORDSPERBLOCK / 2; i < RECORDSPERBLOCK; i++){
00689             (currentBlock->getNextBlock())->addRecord(recordBlock[i].get_field("zip"));
00690             currentBlock->deleteRecord( recordBlock[i].get_field("zip") );
00691         }
00692         //add the new record to the block
00693         currentBlock->addRecord( record.get_field("zip") );
00694         blockCount++;
00695         break; //stop searching through linked list of blocks
00696     }
00697     else{ //if a current empty block doesn't exist, create a new block.....
00698         Block* newBlockPtr = new Block;
00699         newBlockPtr->setRBN(blockCount);
00700         newBlockPtr->setNextBlock( currentBlock->getNextBlock() );
00701         newBlockPtr->setPrevBlock( (currentBlock->getNextBlock())->getPreviousBlock() );
00702         (currentBlock->getNextBlock())->setPrevBlock(newBlockPtr);
00703         currentBlock->setNextBlock(newBlockPtr);
00704         //split the data into the new block number
00705         currentBlock->getRecords( recordBlock ); //get the pkeys
00706         for(int i = RECORDSPERBLOCK / 2; i < RECORDSPERBLOCK; i++){
00707             (currentBlock->getNextBlock())->addRecord(recordBlock[i].get_field("zip"));
00708             currentBlock->deleteRecord( recordBlock[i].get_field("zip") );
00709         }
00710         //add the new record to the block
00711         currentBlock->addRecord( record.get_field("zip") );
00712         blockCount++;
00713         break; //stop searching through linked list of blocks
00714     }
00715 }
00716 else{
00717     currentBlock->addRecord( record.get_field("zip") );
00718 }
00719     break; //stop searching through linked list of blocks
00720 }
00721 else{
00722     currentBlock = currentBlock->getNextBlock();
00723 }
00724 }
00725
00726 //add record to us_postal_codes.txt
00727 fstream recordAvailList;
00728 string str = "";
00729 string strTemp = "";
00730 string offset = "";
00731 string position = "";
00732 recordAvailList.open(recordAvailListFileName);
00733 if( recordAvailList.peek() != EOF ){ //if recordAvailList is not empty
00734     fstream usPostalCodes;
00735     usPostalCodes.open("us_postal_codes.txt");
00736     getline(recordAvailList, str); //get the offset and vector position from avail list
00737     int i = 0;
00738     while( str[i] != '/' ){ //parse the offset from the string
00739         offset += str[i];
00740         cout << offset << endl;
00741         i++;
00742     }
00743     i++;
00744     while( i < str.length() ){ //parse the position from the string
00745         position += str[i];
00746         cout << position << endl;
00747         i++;
00748     }
00749     writeToTxt(record, offset, "us_postal_codes.txt");
00750     usPostalCodes.close();
00751     recordAvailList.close();
00752     //update the recordAvailList
00753     recordAvailList.open(recordAvailListFileName);
00754     str += "\n";
00755     cout << str << " str to delete" << endl;
00756     while(recordAvailList.peek() != EOF){
00757         strTemp += recordAvailList.get();
00758         cout << strTemp << endl;
00759         if(strTemp == str){
00760             strTemp = "";
00761         }
00762     }
00763     recordAvailList.close();
00764     remove("availRecordList.txt");
00765     ofstream recordAvailListOut;
00766     recordAvailListOut.open(recordAvailListFileName, ios::app);
00767     cout << strTemp << " result" << endl;
00768     recordAvailListOut << strTemp;
00769     recordAvailListOut.close();
00770     //add record to index vector

```

```

00771     for(int i=0; i<20; ++i)
00772         std::cout << pKeyIndex[i] << ' ';
00773     pKeyIndex.insert(pKeyIndex.begin() + stoi( position ), stoi( record.get_field("zip") ) );
00774     offsetIndex.insert(offsetIndex.begin() + stoi( position ), stoi( offset ) );
00775     cout << endl;
00776     for(int i=0; i<20; ++i)
00777         std::cout << pKeyIndex[i] << ' ';
00778 }
00779 else{ //if recordAvailList is empty
00780     unsigned int nextOffset = offsetIndex.back() + 95; //95 is record length+1
00781     cout << nextOffset << " nextoffset" << endl;
00782     pKeyIndex.push_back( stoi( record.get_field("zip") ) );
00783     offsetIndex.push_back( nextOffset );
00784     writeToTxt(record, to_string( nextOffset ), "us_postal_codes.txt");
00785     ofstream usPostalCodes;
00786     usPostalCodes.open("us_postal_codes.txt", ios::app);
00787     usPostalCodes << endl;
00788     usPostalCodes.close();
00789 }
00790 }
00791
00792 void SequenceSet::rewriteSSFile()
00793 {
00794     //rewrite the sequence set file with missing record
00795     remove("Sequence_Set.txt");
00796     ofstream SSFile;
00797     SSFile.open(SSFileName);
00798     SSFile << "Sequence Set File\n";
00799     SSFile.close();
00800     writeBlocks();
00801 }
00802
00803 //write the record to the postal codes file
00804 void SequenceSet::writeToTxt(Record record, string offset, string _fileName)
00805 {
00806     fstream data;
00807     data.open(_fileName);
00808     data.seekg( stoi( offset ) );
00809
00810     string dataString = "";
00811     string totalString = "";
00812
00813     dataString = record.get_field("Zip");
00814     int fieldLength = 6;
00815     for(int i = 0; i < fieldLength - dataString.length(); i++){
00816         totalString += " ";
00817     }
00818     totalString += dataString;
00819
00820     dataString = record.get_field("city");
00821     fieldLength = 31;
00822     totalString += dataString;
00823     for(int i = 0; i < fieldLength - dataString.length(); i++){
00824         totalString += " ";
00825     }
00826
00827     dataString = record.get_field("state");
00828     totalString += dataString;
00829
00830     dataString = record.get_field("county");
00831     fieldLength = 38;
00832     totalString += dataString;
00833     for(int i = 0; i < fieldLength - dataString.length(); i++){
00834         totalString += " ";
00835     }
00836
00837     dataString = record.get_field("long");
00838     while(dataString.length() > 8){
00839         dataString.pop_back();
00840     }
00841     fieldLength = 8;
00842     for(int i = 0; i < fieldLength - dataString.length(); i++){
00843         totalString += " ";
00844     }
00845     totalString += dataString;
00846
00847     dataString = record.get_field("lat");
00848     fieldLength = 9;
00849     while(dataString.length() > 9){
00850         dataString.pop_back();
00851     }
00852     for(int i = 0; i < fieldLength - dataString.length(); i++){
00853         totalString += " ";
00854     }
00855     totalString += dataString;
00856
00857     data << totalString;

```

```

00858
00859     data.close();
00860 }

```

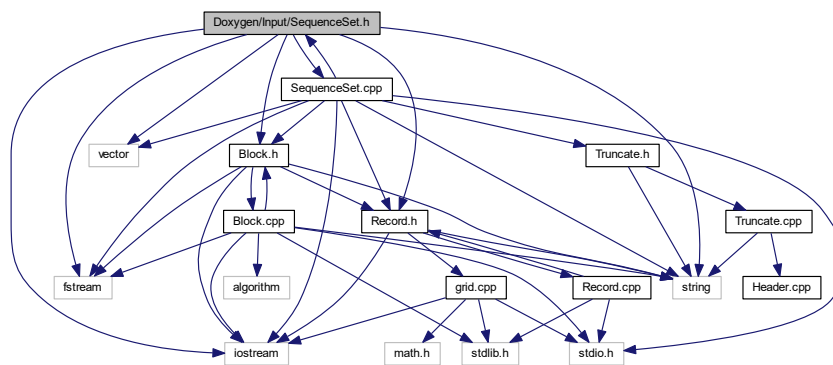
4.19 Doxygen/Input/SequenceSet.h File Reference

```

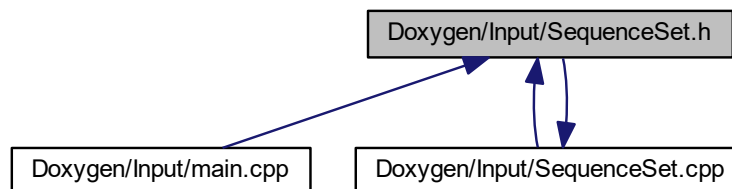
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include "Block.h"
#include "Record.h"
#include "SequenceSet.cpp"

```

Include dependency graph for SequenceSet.h:



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



Classes

- class [SequenceSet](#)

4.20 SequenceSet.h

```

00001
00014 #ifndef SEQUENCESET_H
00015 #define SEQUENCESET_H
00016
00017 #include <iostream>
00018 #include <string>
00019 #include <fstream>
00020 #include <vector>
00021
00022 #include "Block.h"
00023 #include "Record.h"
00024
00025 using namespace std;
00026
00027 class SequenceSet
00028 {
00029 private:
00030     string SSFileName = "Sequence_Set.txt";
00031     string recordAvailListFileName = "availRecordList.txt";
00032     unsigned long long headerLength(string);
00033     unsigned long long blockCount;
00034     unsigned int recordCount;
00035     //unsigned int indexArray[getRecordCount()][2];
00036     vector<unsigned int>pKeyIndex;
00037     vector<unsigned int>offsetIndex;
00038     vector<vector<string>>stateZips;
00039     vector<vector<string>>sKeyCounty;
00040     vector<vector<string>>sKeyPlace;
00041     vector<Block*>blockAvailList;
00042     Record recordBlock[RECORDSPERBLOCK];
00043     Block * headBlock = new Block;
00044
00045     int binarySearchSS(string x);
00046
00047 public:
00048     SequenceSet();
00049
00055     void makeRecordOffsets(string fileName);
00056
00062     void fillIndex();
00063
00068     void fillRecordBlock(unsigned long long blockID);
00069
00074     void writeBlocks();
00075
00081     Record fillRecord(string RecordString);
00082
00088     unsigned int getRecordCount();
00089
00094     string fetch(string pKey);
00095
00100     string fetch(unsigned int pKey);
00101
00106     void addBlockStateKey(unsigned long long blockID);
00107
00112     void sKeyStateBuilder();
00113
00118     string extremeCoord(string, char);
00119
00124     int test();
00125
00126     bool deleteRecord(int pKey);
00127     void addRecord(Record record);
00128     void rewriteSSFile();
00129     void writeToTxt(Record, string, string);
00130 };
00131
00132 #include "SequenceSet.cpp"
00133
00134 #endif
00135

```

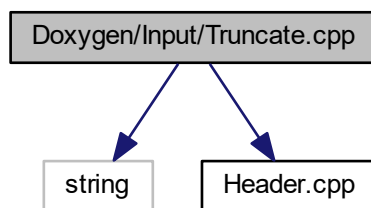
4.21 Doxygen/Input/Truncate.cpp File Reference

```

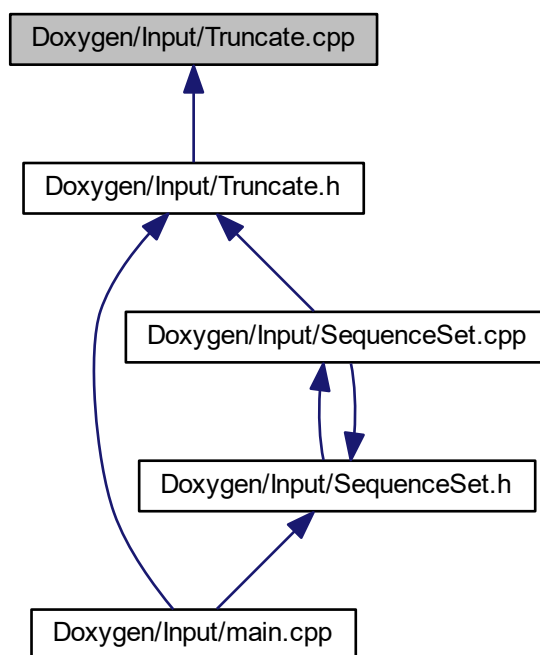
#include <string>
#include "Header.cpp"

```

Include dependency graph for Truncate.cpp:



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



4.21.1 Detailed Description

Author

Christenson, Mark

Definition in file [Truncate.cpp](#).

4.22 Truncate.cpp

```

00001
00006 // #include "Truncate.h"
00007 #include <string>
00008 #include "Header.cpp"
00009
00010 Truncate::Truncate() {
00011     if (true)
00012     {
00013         cout << "Truncate object made";
00014     }
00015 }
00016 Truncate::Truncate(int _size) {
00017     maxLength = _size;
00018 }
00019
00020 string Truncate::truncatedString(string _string) {
00021     string newStr = _string;
00022     newStr.resize(maxLength);
00023
00024     return newStr;
00025 }
00026
00027 string Truncate::modifyString(string & _originalStr) {
00028     _originalStr.resize(maxLength);
00029
00030     return _originalStr;
00031 }

```

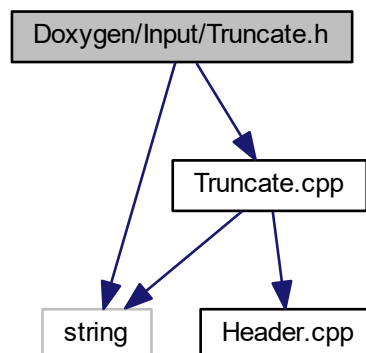
4.23 Doxygen/Input/Truncate.h File Reference

```

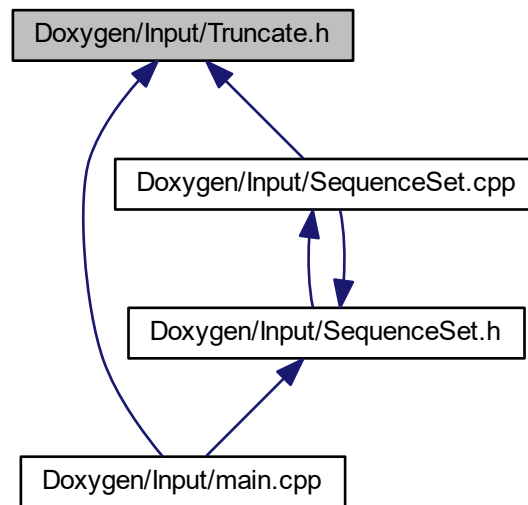
#include <string>
#include "Truncate.cpp"

```

Include dependency graph for Truncate.h:



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



Classes

- class [Truncate](#)

4.24 Truncate.h

```

00001 #ifndef TRUNCATE_H
00002 #define TRUNCATE_H
00003 #include <string>
00004
00005 using namespace std;
00006
00007 class Truncate {
00008 public:
00012     Truncate();
00016     Truncate(int);
00020     string modifyString(string&);
00025     string truncatedString(string);
00026 private:
00027     int maxLength = 10;
00028 };
00029
00030 #include "Truncate.cpp"
00031
00032 #endif
  
```

Index

- addBlockStateKey
 - SequenceSet, [26](#)
- addRecord
 - Block, [7](#)
 - SequenceSet, [27](#)
- binarySearch
 - Block.cpp, [40](#)
- binarySearchSS
 - SequenceSet.cpp, [71](#)
- Block, [5](#)
 - addRecord, [7](#)
 - Block, [6](#), [7](#)
 - blockData, [8](#)
 - deleteRecord, [8](#)
 - getLastRecordPKey, [9](#)
 - getNextBlock, [9](#)
 - getPreviousBlock, [10](#)
 - getRBN, [10](#)
 - getRecordCount, [10](#)
 - getRecords, [11](#)
 - search, [12](#)
 - setNextBlock, [12](#)
 - setPrevBlock, [13](#)
 - setRBN, [13](#)
 - write, [14](#)
- Block.cpp
 - binarySearch, [40](#)
 - convertIntArrToStrArr, [41](#)
 - convertStrArrToIntArr, [41](#)
 - NULL_INT, [42](#)
 - null_str, [42](#)
- blockData
 - Block, [8](#)
- BLOCKFILLCOUNT
 - Header.cpp, [52](#)
- BLOCKLENGTH
 - Header.cpp, [52](#)
- blockTester
 - main.cpp, [54](#)
- CITY
 - Record_Test_Driver.cpp, [69](#)
- convertIntArrToStrArr
 - Block.cpp, [41](#)
- convertStrArrToIntArr
 - Block.cpp, [41](#)
- COUNTY
 - Record_Test_Driver.cpp, [69](#)
- DATAFILENAME
 - Header.cpp, [52](#)
- DEBUG
 - Header.cpp, [52](#)
- deleteRecord
 - Block, [8](#)
 - SequenceSet, [28](#)
- display
 - Record, [21](#), [22](#)
- Doxygen/Input/Block.cpp, [39](#), [43](#)
- Doxygen/Input/Block.h, [47](#), [48](#)
- Doxygen/Input/grid.cpp, [49](#)
- Doxygen/Input/Header.cpp, [51](#), [53](#)
- Doxygen/Input/main.cpp, [54](#), [58](#)
- Doxygen/Input/Record.cpp, [60](#), [61](#)
- Doxygen/Input/Record.h, [66](#), [67](#)
- Doxygen/Input/Record_Test_Driver.cpp, [67](#), [69](#)
- Doxygen/Input/SequenceSet.cpp, [70](#), [72](#)
- Doxygen/Input/SequenceSet.h, [82](#), [83](#)
- Doxygen/Input/Truncate.cpp, [83](#), [85](#)
- Doxygen/Input/Truncate.h, [85](#), [86](#)
- extremeCoord
 - SequenceSet, [29](#)
- fetch
 - SequenceSet, [30](#)
- Field
 - Record_Test_Driver.cpp, [68](#)
- fillIndex
 - SequenceSet, [31](#)
- FILLPERCENT
 - Header.cpp, [52](#)
- fillRecord
 - SequenceSet, [31](#)
- fillRecordBlock
 - SequenceSet, [32](#)
- G
 - Record_Test_Driver.cpp, [69](#)
- get_field
 - Record, [22](#)
- getDistance
 - Grid, [16](#)
- getLastRecordPKey
 - Block, [9](#)
- getLatitude
 - Grid, [16](#)
- getLongitude
 - Grid, [17](#)

- getNextBlock
 - Block, [9](#)
- getPreviousBlock
 - Block, [10](#)
- getRBN
 - Block, [10](#)
- getRecordCount
 - Block, [10](#)
 - SequenceSet, [32](#)
- getRecords
 - Block, [11](#)
- GRID
 - Record_Test_Driver.cpp, [69](#)
- Grid, [15](#)
 - getDistance, [16](#)
 - getLatitude, [16](#)
 - getLongitude, [17](#)
 - Grid, [15](#)
 - setLatitude, [18](#)
 - setLongitude, [18](#), [19](#)
- Header.cpp
 - BLOCKFILLCOUNT, [52](#)
 - BLOCKLENGTH, [52](#)
 - DATAFILENAME, [52](#)
 - DEBUG, [52](#)
 - FILLPERCENT, [52](#)
 - HEADERENDSTRING, [53](#)
 - RBNLENGTH, [53](#)
 - RECORDSPERBLOCK, [53](#)
 - ZIPLength, [53](#)
- HEADERENDSTRING
 - Header.cpp, [53](#)
- main
 - main.cpp, [55](#)
 - Record_Test_Driver.cpp, [69](#)
- main.cpp
 - blockTester, [54](#)
 - main, [55](#)
 - nullblockTester, [55](#)
 - recordTester, [56](#)
 - SSDeleteAndAddRecordTester, [56](#)
 - truncateTester, [57](#)
- makeRecordOffsets
 - SequenceSet, [33](#)
- modifyString
 - Truncate, [37](#)
- NULL_INT
 - Block.cpp, [42](#)
- null_str
 - Block.cpp, [42](#)
- nullblockTester
 - main.cpp, [55](#)
- P
 - Record_Test_Driver.cpp, [69](#)
- PLACE_NAME
 - Record_Test_Driver.cpp, [69](#)
- RBNLENGTH
 - Header.cpp, [53](#)
- Record, [19](#)
 - display, [21](#), [22](#)
 - get_field, [22](#)
 - Record, [20](#), [21](#)
 - set_field, [23](#)
 - set_grid_point, [23](#)
 - set_longitude_latitude, [24](#)
- Record_Test_Driver.cpp
 - CITY, [69](#)
 - COUNTY, [69](#)
 - Field, [68](#)
 - G, [69](#)
 - GRID, [69](#)
 - main, [69](#)
 - P, [69](#)
 - PLACE_NAME, [69](#)
 - STATE, [69](#)
 - Z, [69](#)
 - ZIP, [69](#)
- RECORDSPERBLOCK
 - Header.cpp, [53](#)
- recordTester
 - main.cpp, [56](#)
- rewriteSSFile
 - SequenceSet, [33](#)
- search
 - Block, [12](#)
- SequenceSet, [25](#)
 - addBlockStateKey, [26](#)
 - addRecord, [27](#)
 - deleteRecord, [28](#)
 - extremeCoord, [29](#)
 - fetch, [30](#)
 - fillIndex, [31](#)
 - fillRecord, [31](#)
 - fillRecordBlock, [32](#)
 - getRecordCount, [32](#)
 - makeRecordOffsets, [33](#)
 - rewriteSSFile, [33](#)
 - SequenceSet, [26](#)
 - sKeyStateBuilder, [33](#)
 - test, [34](#)
 - writeBlocks, [34](#)
 - writeToTxt, [35](#)
- SequenceSet.cpp
 - binarySearchSS, [71](#)
- set_field
 - Record, [23](#)
- set_grid_point
 - Record, [23](#)
- set_longitude_latitude
 - Record, [24](#)
- setLatitude
 - Grid, [18](#)

- setLongitude
 - Grid, [18](#), [19](#)
- setNextBlock
 - Block, [12](#)
- setPrevBlock
 - Block, [13](#)
- setRBN
 - Block, [13](#)
- sKeyStateBuilder
 - SequenceSet, [33](#)
- SSDeleteAndAddRecordTester
 - main.cpp, [56](#)
- STATE
 - Record_Test_Driver.cpp, [69](#)
- test
 - SequenceSet, [34](#)
- Truncate, [36](#)
 - modifyString, [37](#)
 - Truncate, [36](#)
 - truncatedString, [37](#)
- truncatedString
 - Truncate, [37](#)
- truncateTester
 - main.cpp, [57](#)
- write
 - Block, [14](#)
- writeBlocks
 - SequenceSet, [34](#)
- writeToTxt
 - SequenceSet, [35](#)
- Z
 - Record_Test_Driver.cpp, [69](#)
- ZIP
 - Record_Test_Driver.cpp, [69](#)
- ZIPLength
 - Header.cpp, [53](#)