

ecet4640-lab4

1.0

Generated by Doxygen 1.9.6



<b>1 ecet4640-lab4</b>	<b>1</b>
1.1 Intro	1
1.2 Overview	1
1.3 Arguments for program	1
<b>2 Compilation</b>	<b>3</b>
2.1 Compilation Pipelines	3
2.2 Compiling and Running	3
2.3 Screenshot of Compilation	4
2.4 Cleaning	4
<b>3 Data Structure Index</b>	<b>5</b>
3.1 Data Structures	5
<b>4 File Index</b>	<b>7</b>
4.1 File List	7
<b>5 Data Structure Documentation</b>	<b>9</b>
5.1 map Struct Reference	9
5.1.1 Detailed Description	9
5.1.2 Field Documentation	10
5.1.2.1 buckets	10
5.1.2.2 size	10
5.2 map_result Struct Reference	10
5.2.1 Detailed Description	10
5.2.2 Field Documentation	10
5.2.2.1 data	11
5.2.2.2 found	11
5.3 Student Struct Reference	11
5.3.1 Detailed Description	11
5.3.2 Field Documentation	12
5.3.2.1 active	12
5.3.2.2 age	12
5.3.2.3 fullName	12
5.3.2.4 gpa	12
5.3.2.5 lastLogin	12
5.3.2.6 loginDuration	13
5.3.2.7 userID	13
<b>6 File Documentation</b>	<b>15</b>
6.1 src/server/Build.c File Reference	15
6.1.1 Detailed Description	16
6.1.2 Function Documentation	16
6.1.2.1 BuildStudentMap()	16

6.1.2.2 CalculateCumulative()	17
6.1.2.3 PopulateStudents()	18
6.1.2.4 ProcessWhoLine()	18
6.1.2.5 ReadACP()	19
6.1.2.6 ReadAcpPipeLine()	20
6.1.2.7 ReadCumulativeFileLine()	21
6.1.2.8 ReadInitialCumulative()	22
6.1.2.9 SetAllStudentsInactive()	23
6.1.2.10 UpdateFromWho()	24
6.1.2.11 WriteStudentsToMemory()	24
6.1.3 Variable Documentation	25
6.1.3.1 dirty	25
6.1.3.2 students	25
6.2 Build.c	26
6.3 src/server/Build.h File Reference	28
6.3.1 Detailed Description	30
6.3.2 Function Documentation	30
6.3.2.1 BuildStudentMap()	30
6.3.2.2 CalculateCumulative()	31
6.3.2.3 PopulateStudents()	32
6.3.2.4 ProcessWhoLine()	32
6.3.2.5 ReadACP()	33
6.3.2.6 ReadAcpPipeLine()	34
6.3.2.7 ReadCumulativeFileLine()	35
6.3.2.8 ReadInitialCumulative()	36
6.3.2.9 SetAllStudentsInactive()	37
6.3.2.10 UpdateFromWho()	38
6.3.2.11 WriteStudentsToMemory()	38
6.3.3 Variable Documentation	39
6.3.3.1 dirty	39
6.3.3.2 students	39
6.4 Build.h	40
6.5 src/server/Data.c File Reference	40
6.5.1 Detailed Description	41
6.5.2 Variable Documentation	41
6.5.2.1 Data_IDs	41
6.5.2.2 Data_Names	41
6.6 Data.c	42
6.7 src/server/Data.h File Reference	42
6.7.1 Detailed Description	43
6.7.2 Macro Definition Documentation	43
6.7.2.1 DATA_ID_MAX_LENGTH	44

6.7.2.2 DATA_NAME_MAX_LENGTH . . . . .	44
6.7.2.3 DATA_NUM_RECORDS . . . . .	44
6.7.2.4 DATA_SIZE . . . . .	44
6.7.3 Variable Documentation . . . . .	44
6.7.3.1 Data_IDs . . . . .	44
6.7.3.2 Data_Names . . . . .	45
6.8 Data.h . . . . .	45
6.9 src/server/Files.c File Reference . . . . .	45
6.9.1 Detailed Description . . . . .	46
6.9.2 Function Documentation . . . . .	46
6.9.2.1 CreateInitialCumulativeFile() . . . . .	46
6.9.2.2 CreateInitialUserDataFile() . . . . .	47
6.9.2.3 CreateLockfile() . . . . .	48
6.9.2.4 DeleteLockfile() . . . . .	49
6.9.2.5 DoesLockfileExist() . . . . .	50
6.9.2.6 FileExists() . . . . .	50
6.9.2.7 FillStudentMapFromFile() . . . . .	51
6.9.2.8 WriteStudentArrayToFile() . . . . .	52
6.10 Files.c . . . . .	53
6.11 src/server/Files.h File Reference . . . . .	54
6.11.1 Detailed Description . . . . .	56
6.11.2 Macro Definition Documentation . . . . .	56
6.11.2.1 LOCKFILE . . . . .	56
6.11.2.2 STATIC_USER_CUMULATIVE_FILE . . . . .	57
6.11.2.3 STATIC_USER_DATA_FILE . . . . .	57
6.11.3 Function Documentation . . . . .	57
6.11.3.1 CreateInitialCumulativeFile() . . . . .	57
6.11.3.2 CreateInitialUserDataFile() . . . . .	58
6.11.3.3 CreateLockfile() . . . . .	59
6.11.3.4 DeleteLockfile() . . . . .	60
6.11.3.5 DoesLockfileExist() . . . . .	61
6.11.3.6 FileExists() . . . . .	61
6.11.3.7 FillStudentMapFromFile() . . . . .	62
6.11.3.8 WriteStudentArrayToFile() . . . . .	63
6.12 Files.h . . . . .	64
6.13 src/server/main.c File Reference . . . . .	64
6.13.1 Detailed Description . . . . .	65
6.13.2 Function Documentation . . . . .	65
6.13.2.1 main() . . . . .	65
6.14 main.c . . . . .	66
6.15 src/server/map.c File Reference . . . . .	67
6.15.1 Detailed Description . . . . .	68

6.15.2 Function Documentation	68
6.15.2.1 _bucket_delete()	68
6.15.2.2 _bucket_get()	68
6.15.2.3 _bucket_insert()	69
6.15.2.4 hash_log2()	69
6.15.2.5 hash_string()	70
6.15.2.6 hash_upperLimit()	70
6.15.2.7 Map_Delete()	71
6.15.2.8 Map_Get()	72
6.15.2.9 Map_Set()	73
6.15.2.10 NewMap()	74
6.15.3 Variable Documentation	75
6.15.3.1 char_ratio	75
6.16 map.c	75
6.17 src/server/map.h File Reference	77
6.17.1 Detailed Description	78
6.17.2 Function Documentation	79
6.17.2.1 hash_log2()	79
6.17.2.2 hash_string()	79
6.17.2.3 hash_upperLimit()	80
6.17.2.4 Map_Delete()	81
6.17.2.5 Map_Get()	82
6.17.2.6 Map_Set()	83
6.17.2.7 NewMap()	84
6.18 map.h	85
6.19 src/server/memShare.c File Reference	86
6.19.1 Detailed Description	87
6.19.2 Function Documentation	87
6.19.2.1 CreateSharedMemory()	87
6.19.2.2 DestroySharedMemory()	87
6.19.2.3 GetMemoryPointer()	88
6.19.2.4 ReleaseMemoryPointer()	88
6.20 memShare.c	89
6.21 src/server/memShare.h File Reference	89
6.21.1 Detailed Description	91
6.21.2 Macro Definition Documentation	91
6.21.2.1 MEM_KEY	91
6.21.2.2 MEM_PERMISSIONS	91
6.21.2.3 MEM_SIZE	91
6.21.3 Function Documentation	91
6.21.3.1 CreateSharedMemory()	92
6.21.3.2 DestroySharedMemory()	92

6.21.3.3 GetMemoryPointer()	92
6.21.3.4 ReleaseMemoryPointer()	93
6.22 memShare.h	94
6.23 src/server/Process.c File Reference	94
6.23.1 Detailed Description	95
6.23.2 Function Documentation	95
6.23.2.1 HelpCommand()	95
6.23.2.2 IndicateRereadDone()	96
6.23.2.3 IndicateRereadNeeded()	96
6.23.2.4 Initialize()	97
6.23.2.5 IsRereadNeeded()	98
6.23.2.6 Process()	98
6.23.2.7 ResetCommand()	99
6.23.2.8 RunCommand()	100
6.23.2.9 RunHeadless()	101
6.23.2.10 SignalHandle()	102
6.23.2.11 StopCommand()	103
6.23.2.12 TerminateExistingServer()	104
6.23.3 Variable Documentation	104
6.23.3.1 initial_cumulative_times	104
6.23.3.2 is_stopping	104
6.23.3.3 student_map	105
6.24 Process.c	105
6.25 src/server/Process.h File Reference	109
6.25.1 Detailed Description	110
6.25.2 Function Documentation	110
6.25.2.1 HelpCommand()	110
6.25.2.2 IndicateRereadDone()	111
6.25.2.3 IndicateRereadNeeded()	111
6.25.2.4 Initialize()	112
6.25.2.5 IsRereadNeeded()	113
6.25.2.6 Process()	113
6.25.2.7 ResetCommand()	114
6.25.2.8 RunCommand()	115
6.25.2.9 RunHeadless()	116
6.25.2.10 SignalHandle()	117
6.25.2.11 StopCommand()	118
6.25.2.12 TerminateExistingServer()	119
6.25.3 Variable Documentation	119
6.25.3.1 is_stopping	119
6.25.3.2 student_map	119
6.26 Process.h	120

---

6.27 src/server/util.c File Reference . . . . .	120
6.27.1 Detailed Description . . . . .	121
6.27.2 Function Documentation . . . . .	121
6.27.2.1 RandomFlag() . . . . .	121
6.27.2.2 RandomFloat() . . . . .	121
6.27.2.3 RandomInteger() . . . . .	122
6.28 util.c . . . . .	123
6.29 src/server/util.h File Reference . . . . .	123
6.29.1 Detailed Description . . . . .	123
6.29.2 Function Documentation . . . . .	124
6.29.2.1 RandomFlag() . . . . .	124
6.29.2.2 RandomFloat() . . . . .	124
6.29.2.3 RandomInteger() . . . . .	125
6.30 util.h . . . . .	125
<b>Index</b>	<b>127</b>



# Chapter 1

## ecet4640-lab4

### 1.1 Intro

This program reads user information using the `who` command and publishes that information to shared virtual memory for client processes to read. It updates every second.

The [main.c](#) page is a good starting point for following the program control flow.

### 1.2 Overview

The first time the program runs, it generates files containing static user data and the cumulative login times for each user. As the server runs, it will recheck the result of 'who' and 'ac -p' to update the student's cumulative login times, determine which students are actively logged in, and what time they last logged in. This information is stored in a read-shared memory segment so clients can access it using the shared memory key. If necessary, it also updates student information in the file.

Only one server process should be running at a given time. To that end, a running server creates a lockfile in the `/tmp` folder and deletes the lockfile when it is done. New servers will not be started if a lockfile exists, but the running server can be stopped by passing the command line argument 'stop' to the binary. There are other command line arguments available, as detailed below.

### 1.3 Arguments for program

Argument	Description	Calls
help	Prints usage of program.	<a href="#">HelpCommand()</a>
reset	Resets and re-randomizes the static user data and restarts the cumulative time tracking.	<a href="#">ResetCommand()</a>
stop	Stops an existing server process if it is running.	<a href="#">StopCommand()</a>
headless	Runs the program headlessly in the background if it is not already running.	<a href="#">RunHeadless()</a>
run	Runs the server in the current program if it is not already running.	<a href="#">RunCommand()</a>

**Author**

Karl Miller

Paul Shriner

Christian Messmer

## Chapter 2

# Compilation

### 2.1 Compilation Pipelines

There are several compilation pipelines, which are described in more detail in the Makefile comments.

The first is for making and running the regular server process. Calling `make` executes this. It uses the files in `src/server` to generate the binary and runs it. This will output the help for the server command. Executing `make server` will make the server binary without running it.

Second is for making the test client process with `make client`. This uses the files from `src/client`. The client process is not documented as it was not part of the program objective, and to avoid further documentation inflation.

Third is for making the test binary. This compiles the files in `tests` and the files in `src/server`, but excludes `src/main.c` so that `tests/main_test.c` will be the program entry point instead. The tests use `CuTest`. The tests are not documented here in order to not inflate the documentation size any further.

### 2.2 Compiling and Running

1. Copy the .zip file to the server
2. Extract the zip file.
3. Enter the unzipped folder.
4. Run `make server`
5. Run `./server run` to run the server in the shell.
6. Press ctrl-c to stop the server.
7. Run `./server headless` to run the server headlessly using nohup.
8. Run `./server reset` to re-randomize the user data and reset the login times.
9. Run `./server stop` to shut down the server. (You may want to leave it running so clients can connect to it)

## 2.3 Screenshot of Compilation

```
[mil7233@draco1 ecet4640-lab4]$ make server
mkdir -p bin/src/server/
gcc -Wall -Wltests -Wltests/lib -Isrc -Isrc/server -Isrc/client -c src/server/util.c -o bin/src/server/util.c.o
mkdir -p bin/src/server/
gcc -Wall -Wltests -Wltests/lib -Isrc -Isrc/server -Isrc/client -c src/server/memShare.c -o bin/src/server/memShare.c.o
mkdir -p bin/src/server/
gcc -Wall -Wltests -Wltests/lib -Isrc -Isrc/server -Isrc/client -c src/server/Data.c -o bin/src/server/Data.c.o
mkdir -p bin/src/server/
gcc -Wall -Wltests -Wltests/lib -Isrc -Isrc/server -Isrc/client -c src/server/map.c -o bin/src/server/map.c.o
mkdir -p bin/src/server/
gcc -Wall -Wltests -Wltests/lib -Isrc -Isrc/server -Isrc/client -c src/server/main.c -o bin/src/server/main.c.o
mkdir -p bin/src/server/
gcc -Wall -Wltests -Wltests/lib -Isrc -Isrc/server -Isrc/client -c src/server/Process.c -o bin/src/server/Process.c.o
mkdir -p bin/src/server/
gcc -Wall -Wltests -Wltests/lib -Isrc -Isrc/server -Isrc/client -c src/server/Build.c -o bin/src/server/Build.c.o
mkdir -p bin/src/server/
gcc -Wall -Wltests -Wltests/lib -Isrc -Isrc/server -Isrc/client -c src/server/Files.c -o bin/src/server/Files.c.o
mkdir -p bin/src/client/
gcc -Wall -Wltests -Wltests/lib -Isrc -Isrc/server -Isrc/client -c src/client/Print.c -o bin/src/client/Print.c.o
mkdir -p bin/src/client/
gcc -Wall -Wltests -Wltests/lib -Isrc -Isrc/server -Isrc/client -c src/client/GetData.c -o bin/src/client/GetData.c.o
gcc bin/src/server/util.c.o bin/src/server/memShare.c.o bin/src/server/Data.c.o bin/src/server/map.c.o bin/src/server/main.c.o bin/
src/Files.c.o bin/src/client/Print.c.o bin/src/client/GetData.c.o -o server
[mil7233@draco1 ecet4640-lab4]$ ./server run

Running server.
static-user-data.txt does not exist. Creating.
static-user-cumulative-start.txt does not exist. Creating.
Student data retrieved from file.
Shared memory allocated.
Server started.
^CReceived shutdown signal.
Server shutting down.
Server terminated.
[mil7233@draco1 ecet4640-lab4]$ ./server headless
Executing: nohup ./server run & exit
Server running headlessly.
[mil7233@draco1 ecet4640-lab4]$ nohup: appending output to 'nohup.out'

[mil7233@draco1 ecet4640-lab4]$ ./server run

Running server.

Server is already running. Run 'server stop' to shut it down first.
[mil7233@draco1 ecet4640-lab4]$ ./server stop

Stopping server...
Server terminated.
[mil7233@draco1 ecet4640-lab4]$
```

Figure 2.1 Compiling on draco1

## 2.4 Cleaning

There are two clean commands.

`make clean` will clean all .o files and binaries.

`make cleanf` will also remove the files generated on server initialization, such as the cumulative login file and user data file.

## Chapter 3

# Data Structure Index

### 3.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">map</a>	A map. Stores key-value pairs for near constant lookup and insertion time . . . . .	9
<a href="#">map_result</a>	The result of a map retrieval . . . . .	10
<a href="#">Student</a>	The student data type . . . . .	11



## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

src/server/ <a href="#">Build.c</a>	
Definitions for functions that populate data structures . . . . .	15
src/server/ <a href="#">Build.h</a>	
Declarations for functions that populate data structures . . . . .	28
src/server/ <a href="#">Data.c</a>	
Data structures and constants . . . . .	40
src/server/ <a href="#">Data.h</a>	
Declarations of types and macros . . . . .	42
src/server/ <a href="#">Files.c</a>	
Declarations of functions that operate on files. . . . .	45
src/server/ <a href="#">Files.h</a>	
Definitions for functions that operate on files . . . . .	54
src/server/ <a href="#">main.c</a>	
Program entry point . . . . .	64
src/server/ <a href="#">map.c</a>	
Definitions for functions relating to a hash map data structure . . . . .	67
src/server/ <a href="#">map.h</a>	
Definitions for functions that operate on a hash map data structure . . . . .	77
src/server/ <a href="#">memShare.c</a>	
Definitions for functions that operate on a shared memory segment . . . . .	86
src/server/ <a href="#">memShare.h</a>	
Declarations for functions that operate on a shared memory segment . . . . .	89
src/server/ <a href="#">Process.c</a>	
Definitions for functions that manage control flow . . . . .	94
src/server/ <a href="#">Process.h</a>	
Declarations for functions that manage control flow. This module handles the processes that this server might execute. It calls functions from the other modules to realize program changes . . .	109
src/server/ <a href="#">util.c</a>	
Definitions for helper functions . . . . .	120
src/server/ <a href="#">util.h</a>	
Declarations for helper functions . . . . .	123





## Chapter 5

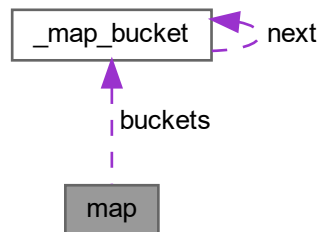
# Data Structure Documentation

### 5.1 map Struct Reference

A map. Stores key-value pairs for near constant lookup and insertion time.

```
#include <map.h>
```

Collaboration diagram for map:



#### Data Fields

- int [size](#)
- struct `_map_bucket` \* [buckets](#)

#### 5.1.1 Detailed Description

A map. Stores key-value pairs for near constant lookup and insertion time.

##### Note

Use `NewMap` to create a new map.

Use `Map_Set` to set a key in the map.

Use `Map_Get` to get a value from the map.

The values stored are of type void pointer.

Definition at line 97 of file [map.h](#).

## 5.1.2 Field Documentation

### 5.1.2.1 buckets

```
struct _map_bucket* buckets
```

Definition at line 102 of file [map.h](#).

### 5.1.2.2 size

```
int size
```

Definition at line 100 of file [map.h](#).

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

- [src/server/map.h](#)

## 5.2 map\_result Struct Reference

The result of a map retrieval.

```
#include <map.h>
```

### Data Fields

- short [found](#)
- void \* [data](#)

### 5.2.1 Detailed Description

The result of a map retrieval.

Definition at line 108 of file [map.h](#).

### 5.2.2 Field Documentation

#### 5.2.2.1 data

```
void* data
```

Definition at line 113 of file [map.h](#).

#### 5.2.2.2 found

```
short found
```

Definition at line 111 of file [map.h](#).

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

- [src/server/map.h](#)

## 5.3 Student Struct Reference

The student data type.

```
#include <Data.h>
```

### Data Fields

- char [userID](#) [[DATA\\_ID\\_MAX\\_LENGTH](#)]  
*The unique user ID.*
- char [fullName](#) [[DATA\\_NAME\\_MAX\\_LENGTH](#)]  
*The user's full name.*
- short [age](#)  
*The user's age (randomized).*
- float [gpa](#)  
*The user's gpa (randomized).*
- short [active](#)  
*Whether the user is currently logged in (1) or not (0).*
- time\_t [lastLogin](#)  
*The last time the user logged in.*
- int [loginDuration](#)  
*The cumulative time the user has been logged in since the server process started.*

### 5.3.1 Detailed Description

The student data type.

Definition at line 45 of file [Data.h](#).

## 5.3.2 Field Documentation

### 5.3.2.1 active

```
short active
```

Whether the user is currently logged in (1) or not (0).

Definition at line 56 of file [Data.h](#).

### 5.3.2.2 age

```
short age
```

The user's age (randomized).

Definition at line 52 of file [Data.h](#).

### 5.3.2.3 fullName

```
char fullName[DATA_NAME_MAX_LENGTH]
```

The user's full name.

Definition at line 50 of file [Data.h](#).

### 5.3.2.4 gpa

```
float gpa
```

The user's gpa (randomized).

Definition at line 54 of file [Data.h](#).

### 5.3.2.5 lastLogin

```
time_t lastLogin
```

The last time the user logged in.

Definition at line 58 of file [Data.h](#).

#### 5.3.2.6 loginDuration

```
int loginDuration
```

The cumulative time the user has been logged in since the server process started.

Definition at line 60 of file [Data.h](#).

#### 5.3.2.7 userID

```
char userID[DATA_ID_MAX_LENGTH]
```

The unique user ID.

Definition at line 48 of file [Data.h](#).

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

- [src/server/Data.h](#)



## Chapter 6

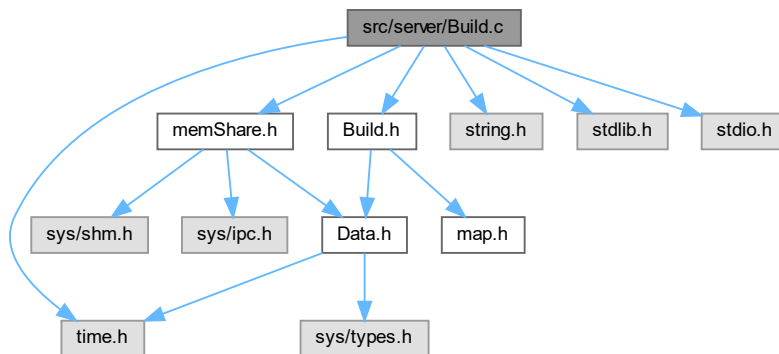
# File Documentation

### 6.1 src/server/Build.c File Reference

Definitions for functions that populate data structures.

```
#include "Build.h"  
#include "memShare.h"  
#include <string.h>  
#include <stdlib.h>  
#include <stdio.h>  
#include <time.h>
```

Include dependency graph for Build.c:



### Functions

- void `PopulateStudents` (char \*\*studentIDs, char \*\*studentNames, int arsize)
- void `BuildStudentMap` (map \*stmap, Student \*studentArr, int studentArrLength)
- int `UpdateFromWho` (map \*stmap)
- int `ProcessWhoLine` (map \*stmap, char \*whoLine, int whoLineLength)
- void `SetAllStudentsInactive` (Student \*stud\_arr, int arr\_len)
- void `WriteStudentsToMemory` (void \*mem\_ptr, Student \*stud\_arr, int arr\_len)
- int `ReadInitialCumulative` (map \*time\_map, char \*filename)
- int `ReadACP` (map \*st\_map)
- void `ReadCumulativeFileLine` (map \*cum\_map, char \*acp\_line)
- int `ReadAcpPipeLine` (map \*stmap, char \*acp\_line)
- void `CalculateCumulative` (Student \*stud\_arr, int stud\_arr\_len, map \*cum\_map)

## Variables

- [Student](#) \* [students](#)
- short [dirty](#) = 1

### 6.1.1 Detailed Description

Definitions for functions that populate data structures.

Definition in file [Build.c](#).

### 6.1.2 Function Documentation

#### 6.1.2.1 BuildStudentMap()

```
void BuildStudentMap (
    map * stmap,
    Student * studentArr,
    int studentArrLength )
```

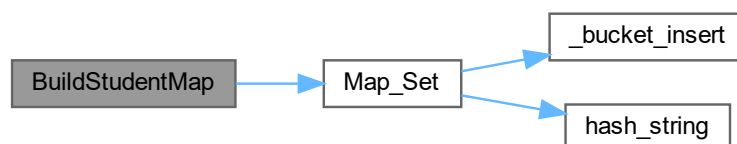
Given a student array, populates a student map, where the student IDs are the key, and the values are pointers to items in the array.

##### Parameters

<i>map</i>	The map structure to populate.
<i>studentArr</i>	An array of student structures.
<i>studentArrLength</i>	The length of the students array.

Definition at line 27 of file [Build.c](#).

Here is the call graph for this function:





Here is the caller graph for this function:



### 6.1.2.2 CalculateCumulative()

```
void CalculateCumulative (
    Student * stud_arr,
    int stud_arr_len,
    map * time_map )
```

Calculates the cumulative time for each student by subtracting `cum_map[studentID]` from `student.loginDuration`.

#### Warning

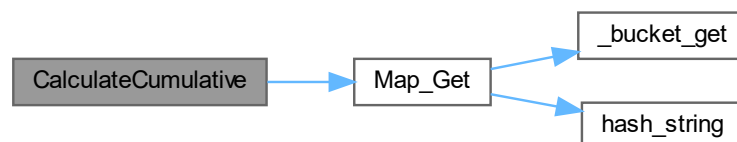
each `student.loginDuration` must have already been set to the total cumulative time logged in.

#### Parameters

<i>stud_arr</i>	The student's array.
<i>arr_len</i>	The length of students array.
<i>time_map</i>	A map mapping studentIDs to their cumulative login time when the server was started.

Definition at line 205 of file [Build.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.1.2.3 PopulateStudents()

```

void PopulateStudents (
    char ** studentIDs,
    char ** studentNames,
    int arsize )
  
```

Allocate and populate the Students array with data.

#### Parameters

<i>studentIDs</i>	An array of student IDs.
<i>studentNames</i>	An array of student names.
<i>arsize</i>	The size of the array to allocate.

#### Warning

studentIDs and studentNames must both be arsize in length.

Definition at line 16 of file [Build.c](#).

Here is the caller graph for this function:



### 6.1.2.4 ProcessWhoLine()

```

int ProcessWhoLine (
    map * stmap,
    char * whoLine,
    int whoLineLength )
  
```

Processes a single line as read from the 'who' shell command. Uses that data to update the relevant student by retrieving them from the student map. Updates that students last login time. Also sets 'active' to 1 for the found student.

**Attention**

May set dirty to 1.

**Parameters**

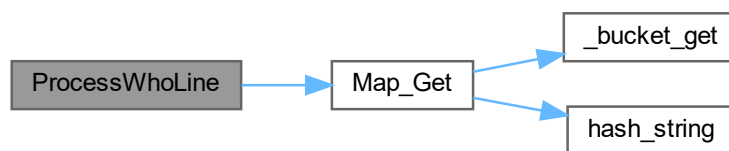
<i>stmap</i>	The student map.
<i>whoLine</i>	The line of text, such as returned from fgets
<i>whoLineLength</i>	The length of that text.

**Returns**

0 if success, -1 if the student was not found in the map.

Definition at line 60 of file [Build.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**6.1.2.5 ReadACP()**

```
int ReadACP (
    map * st_map )
```

Pipes `ac -p`, then calls `ReadCumulativeLine` to update the student map.

**Note**

After this runs, the student map cumulative will be their total login time in the system. This total time must be subtracted from the cumulative map time to find the time they have been logged in since the program started.

**Parameters**

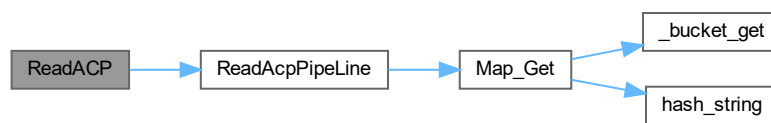
<i>st_map</i>	The students map.
---------------	-------------------

**Returns**

0 on success.

Definition at line 152 of file [Build.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**6.1.2.6 ReadAcpPipeLine()**

```

int ReadAcpPipeLine (
    map * stmap,
    char * acp_line )

```

Reads a single line from the result of `ac -p` into the students map.

**Parameters**

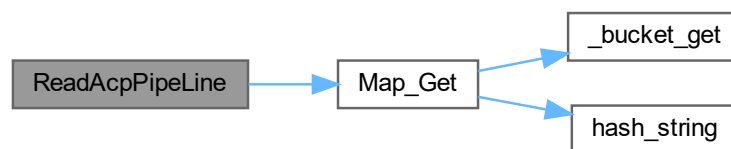
<i>stmap</i>	A map of students.
<i>acp_line</i>	A string representing 1 line result from <code>ac -p</code> .

**Returns**

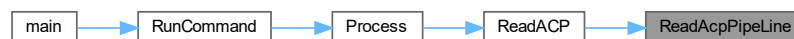
-1 if `acp_line` is NULL or length is less than 1, otherwise 0.

Definition at line 186 of file [Build.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**6.1.2.7 ReadCumulativeFileLine()**

```

void ReadCumulativeFileLine (
    map * time_map,
    char * acp_line )
  
```

Reads a single line from the initial cumulative file and updates the map so that userID maps to a float value in the initial file.

**Note**

A line is structured like this: `mes08346 10.06` It finishes with a line starting with `total`; this line should be disregarded.

**Parameters**

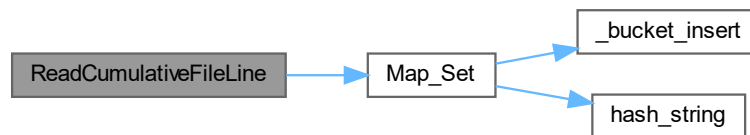
<i>time_map</i>	The cumulative map.
<i>acp_line</i>	A single line from <code>ac -p</code> .

**Returns**

-1 ...

Definition at line 176 of file [Build.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**6.1.2.8 ReadInitialCumulative()**

```

int ReadInitialCumulative (
    map * time_map,
    char * filename )
  
```

Populates the cumulative map by reading from the initial cumulative file. The map will be of the form [userID] -> minutes\_float

The map will contain users who we don't care about, but it doesn't matter.

**Parameters**

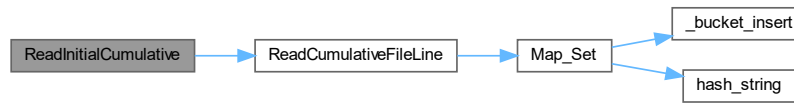
<i>time_map</i>	A map of cumulative times. Different from the students map.
<i>filename</i>	The filename where the initial cumulative times are located.

**Returns**

0 if success. -1 if it failed to find the file.

Definition at line 135 of file [Build.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.1.2.9 SetAllStudentsInactive()

```

void SetAllStudentsInactive (
    Student * stud_arr,
    int arr_len )
  
```

Sets the 'active' property on all students in the students array to 0.

##### Parameters

<i>stud_arr</i>	The students array.
<i>arr_len</i>	The length of the students array.

Definition at line 108 of file [Build.c](#).

Here is the caller graph for this function:



### 6.1.2.10 UpdateFromWho()

```
int UpdateFromWho (
    map * stmap )
```

Executes the 'who' command by reading from a file pipe. Calls ProcessWhoLine for each line, to realize updates in the user data from the who command.

#### Parameters

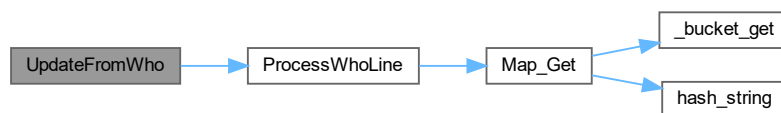
<i>stmap</i>	The student map.
--------------	------------------

#### Returns

0 if succesful, otherwise nonzero.

Definition at line 40 of file [Build.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.1.2.11 WriteStudentsToMemory()

```
void WriteStudentsToMemory (
    void * mem_ptr,
    Student * stud_arr,
    int arr_len )
```

Writes the students array to the location specified by `mem_ptr` (eg. the shared memory segment).



**Parameters**

<i>mem_ptr</i>	The address to write at.
<i>stud_arr</i>	The students array to write.
<i>arr_len</i>	The length of the students array.

Definition at line 117 of file [Build.c](#).

Here is the caller graph for this function:



### 6.1.3 Variable Documentation

#### 6.1.3.1 dirty

```
short dirty = 1
```

Set to '1' if there are changes that should be written to a file.

Definition at line 38 of file [Build.c](#).

#### 6.1.3.2 students

```
Student* students
```

A pointer to the students array. It is heap allocated with malloc, when PopulateStudents is called.

**Note**

Generally this array and its length are passed around via parameters, to decouple as much as possible and enable simple testing and dummy data.

Definition at line 14 of file [Build.c](#).

## 6.2 Build.c

[Go to the documentation of this file.](#)

```

00001
00005 #include "Build.h"
00006 #include "memShare.h"
00007 #include <string.h>
00008 #include <stdlib.h>
00009 #include <stdio.h>
00010 #include <time.h>
00011
00012 // ~~~~~ Data Structures ~~~~~
00013
00014 Student *students;
00015
00016 void PopulateStudents(char **studentIDs, char **studentNames, int arsize)
00017 {
00018     students = malloc(sizeof(Student) * arsize);
00019     int i;
00020     for (i = 0; i < arsize; i++)
00021     {
00022         strcpy(students[i].userID, studentIDs[i]);
00023         strcpy(students[i].fullName, studentNames[i]);
00024     }
00025 }
00026
00027 void BuildStudentMap(map *stmap, Student *studentArr, int studentArrLength)
00028 {
00029     int i;
00030     for (i = 0; i < studentArrLength; i++)
00031     {
00032         Map_Set(stmap, studentArr[i].userID, (void *)(&studentArr[i]));
00033     }
00034 }
00035
00036 // ~~~~~ Processing ~~~~~
00037
00038 short dirty = 1; // start dirty
00039
00040 int UpdateFromWho(map *stmap)
00041 {
00042     char command[4] = "who";
00043     char line[100];
00044     FILE *fpipe;
00045     fpipe = popen(command, "r");
00046     if (fpipe == NULL)
00047     {
00048         return -1;
00049     }
00050
00051     while (fgets(line, sizeof(line), fpipe) != NULL)
00052     {
00053         ProcessWhoLine(stmap, line, strlen(line));
00054     }
00055     pclose(fpipe);
00056
00057     return 0;
00058 }
00059
00060 int ProcessWhoLine(map *stmap, char *whoLine, int whoLineLength)
00061 {
00062     char userId[20];
00063     char dateString[50];
00064     char timeString[20];
00065     int read_total = 0;
00066     int read;
00067     sscanf(whoLine, "%s %n", userId, &read);
00068     read_total += read;
00069
00070     map_result mr = Map_Get(stmap, userId);
00071     if (!mr.found)
00072     { // if we can't find that person in the map, return early
00073         return -1;
00074     }
00075     Student *student = (Student *)mr.data;
00076
00077     sscanf(whoLine + read_total, "%s %n", dateString, &read); // will be thrown away. eg 'pts/1'
00078     read_total += read;
00079     sscanf(whoLine + read_total, "%s %n", timeString, &read); // read the date string
00080     read_total += read;
00081     sscanf(whoLine + read_total, "%s %n", timeString, &read); // read the time string
00082     strcat(dateString, " ");
00083     strcat(dateString, timeString); // concatenate the time string back to the date string
00084
00085     time_t now = time(NULL);

```

```

00086     struct tm dtime = *localtime(&now);
00087     dtime.tm_sec = 0;
00088
00089     memset(&dtime, 0, sizeof(struct tm));
00090
00091     sscanf(dateString, "%d-%d-%d %d:%d", &(dtime.tm_year), &(dtime.tm_mon), &(dtime.tm_mday),
&(dtime.tm_hour), &(dtime.tm_min));
00092
00093     dtime.tm_year -= 1900;
00094     dtime.tm_mon -= 1;
00095     dtime.tm_hour -= 1;
00096
00097     time_t parsed_time = mktime(&dtime);
00098
00099     if (student->lastLogin != parsed_time)
00100     {
00101         student->lastLogin = parsed_time;
00102         dirty = 1;
00103     }
00104     student->active = 1;
00105     return 0;
00106 }
00107
00108 void SetAllStudentsInactive(Student *stud_arr, int arr_len)
00109 {
00110     int i;
00111     for (i = 0; i < arr_len; i++)
00112     {
00113         stud_arr[i].active = 0;
00114     }
00115 }
00116
00117 void WriteStudentsToMemory(void *mem_ptr, Student *stud_arr, int arr_len)
00118 {
00119     Student *memloc = (Student *)mem_ptr;
00120     int i;
00121     for (i = 0; i < arr_len; i++)
00122     {
00123         strcpy(memloc[i].userID, stud_arr[i].userID);
00124         strcpy(memloc[i].fullName, stud_arr[i].fullName);
00125         memloc[i].age = stud_arr[i].age;
00126         memloc[i].gpa = stud_arr[i].gpa;
00127         memloc[i].active = stud_arr[i].active;
00128         memloc[i].lastLogin = stud_arr[i].lastLogin;
00129         memloc[i].loginDuration = stud_arr[i].loginDuration;
00130     }
00131 }
00132
00133 // ~~~~~ Cumulative Processing ~~~~~
00134
00135 int ReadInitialCumulative(map *time_map, char *filename)
00136 {
00137     FILE *file = fopen(filename, "r");
00138     char line[100];
00139     if (file == NULL)
00140     {
00141         return -1;
00142     }
00143     while (fgets(line, sizeof(line), file) != NULL)
00144     {
00145         ReadCumulativeFileLine(time_map, line);
00146     }
00147
00148     fclose(file);
00149     return 0;
00150 }
00151
00152 int ReadACP(map *st_map)
00153 {
00154     char command[6] = "ac -p";
00155     char line[300];
00156     FILE *fpipe;
00157     fpipe = popen(command, "r");
00158     if (fpipe == NULL)
00159     {
00160         return -1;
00161     }
00162     int err;
00163     while (fgets(line, sizeof(line), fpipe) != NULL)
00164     {
00165         err = ReadAcpPipeLine(st_map, line);
00166         if (err)
00167         {
00168             printf("\nError %d reading acp pipeline.", err);
00169             break;
00170         }
00171     }

```

```

00172     pclose(fpipeline);
00173     return 0;
00174 }
00175
00176 void ReadCumulativeFileLine(map *cum_map, char *acp_line)
00177 {
00178     char userId[20];
00179     float minutes;
00180     sscanf(acp_line, "%s %f", userId, &minutes);
00181     // int seconds = (int) (minutes * 60)
00182     long seconds = (long) (minutes * 60);
00183     Map_Set(cum_map, userId, (void *)seconds);
00184 }
00185
00186 int ReadAcpPipeline(map *stmap, char *acp_line)
00187 {
00188     if (acp_line == NULL || strlen(acp_line) < 1)
00189     {
00190         return -1;
00191     }
00192     char userId[40];
00193     float minutes;
00194     sscanf(acp_line, "%s %f", userId, &minutes);
00195     map_result result = Map_Get(stmap, userId);
00196     if (result.found)
00197     {
00198         Student *student = (Student *)result.data;
00199         int seconds = (int) (minutes * 60);
00200         student->loginDuration = seconds;
00201     }
00202     return 0;
00203 }
00204
00205 void CalculateCumulative(Student *stud_arr, int stud_arr_len, map *cum_map)
00206 {
00207     int i;
00208     for (i = 0; i < stud_arr_len; i++)
00209     {
00210         map_result result = Map_Get(cum_map, stud_arr[i].userID);
00211         if (result.found)
00212         {
00213             long time_at_server_start = (long)result.data;
00214             stud_arr[i].loginDuration = stud_arr[i].loginDuration - time_at_server_start;
00215         }
00216     }
00217 }

```

## 6.3 src/server/Build.h File Reference

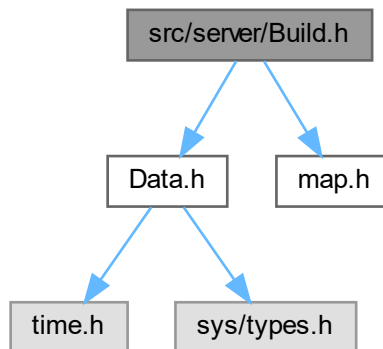
Declarations for functions that populate data structures.

```

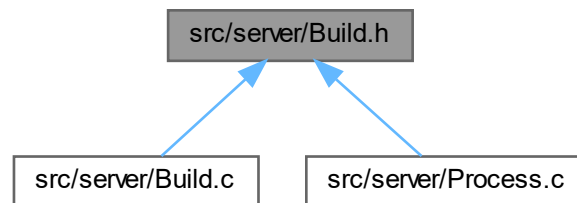
#include "Data.h"
#include "map.h"

```

Include dependency graph for Build.h:



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



## Functions

- void [PopulateStudents](#) (char \*\*studentIDs, char \*\*studentNames, int arsize)
- void [BuildStudentMap](#) ([map](#) \*stmap, [Student](#) \*studentArr, int studentArrLength)
- int [UpdateFromWho](#) ([map](#) \*stmap)
- int [ProcessWhoLine](#) ([map](#) \*stmap, char \*whoLine, int whoLineLength)
- void [SetAllStudentsInactive](#) ([Student](#) \*stud\_arr, int arr\_len)
- void [WriteStudentsToMemory](#) (void \*mem\_ptr, [Student](#) \*stud\_arr, int arr\_len)
- int [ReadInitialCumulative](#) ([map](#) \*time\_map, char \*filename)
- int [ReadACP](#) ([map](#) \*st\_map)
- void [ReadCumulativeFileLine](#) ([map](#) \*time\_map, char \*acp\_line)
- int [ReadAcpPipeLine](#) ([map](#) \*stmap, char \*acp\_line)
- void [CalculateCumulative](#) ([Student](#) \*stud\_arr, int stud\_arr\_len, [map](#) \*time\_map)

## Variables

- [Student](#) \* [students](#)
- short [dirty](#)

### 6.3.1 Detailed Description

Declarations for functions that populate data structures.

Definition in file [Build.h](#).

### 6.3.2 Function Documentation

#### 6.3.2.1 BuildStudentMap()

```
void BuildStudentMap (
    map * stmap,
    Student * studentArr,
    int studentArrLength )
```

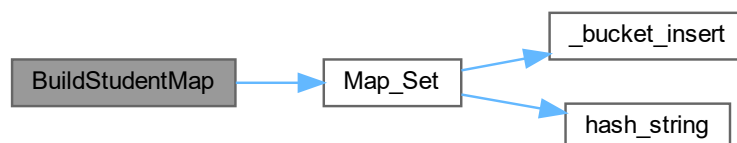
Given a student array, populates a student map, where the student IDs are the key, and the values are pointers to items in the array.

##### Parameters

<i>map</i>	The map structure to populate.
<i>studentArr</i>	An array of student structures.
<i>studentArrLength</i>	The length of the students array.

Definition at line 27 of file [Build.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.3.2.2 CalculateCumulative()

```
void CalculateCumulative (
    Student * stud_arr,
    int stud_arr_len,
    map * time_map )
```

Calculates the cumulative time for each student by subtracting cum\_map[studentID] from student.loginDuration.

#### Warning

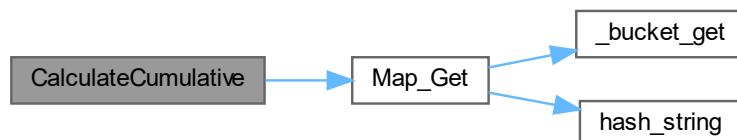
each student.loginDuration must have already been set to the total cumulative time logged in.

#### Parameters

<i>stud_arr</i>	The student's array.
<i>arr_len</i>	The length of students array.
<i>time_map</i>	A map mapping studentIds to their cumulative login time when the server was started.

Definition at line 205 of file [Build.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.3.2.3 PopulateStudents()

```
void PopulateStudents (
    char ** studentIDs,
    char ** studentNames,
    int arsize )
```

Allocate and populate the Students array with data.

#### Parameters

<i>studentIDs</i>	An array of student IDs.
<i>studentNames</i>	An array of student names.
<i>arsize</i>	The size of the array to allocate.

#### Warning

*studentIDs* and *studentNames* must both be *arsize* in length.

Definition at line 16 of file [Build.c](#).

Here is the caller graph for this function:



### 6.3.2.4 ProcessWhoLine()

```
int ProcessWhoLine (
    map * stmap,
    char * whoLine,
    int whoLineLength )
```

Processes a single line as read from the 'who' shell command. Uses that data to update the relevant student by retrieving them from the student map. Updates that students last login time. Also sets 'active' to 1 for the found student.

#### Attention

May set dirty to 1.

#### Parameters

<i>stmap</i>	The student map.
<i>whoLine</i>	The line of text, such as returned from <code>fgets</code>
<i>whoLineLength</i>	The length of that text.

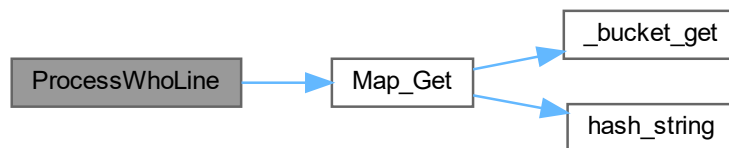


**Returns**

0 if success, -1 if the student was not found in the map.

Definition at line 60 of file [Build.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**6.3.2.5 ReadACP()**

```
int ReadACP (
    map * st_map )
```

Pipes ac -p, then calls `ReadCumulativeLine` to update the student map.

**Note**

After this runs, the student map cumulative will be their total login time in the system. This total time must be subtracted from the cumulative map time to find the time they have been logged in since the program started.

**Parameters**

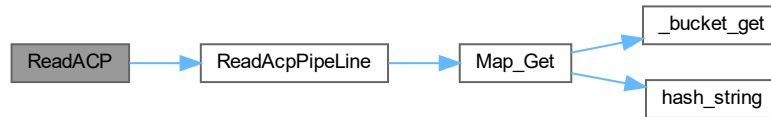
<code>st_map</code>	The students map.
---------------------	-------------------

**Returns**

0 on success.

Definition at line 152 of file [Build.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.3.2.6 ReadAcpPipeLine()

```

int ReadAcpPipeLine (
    map * stmap,
    char * acp_line )
  
```

Reads a single line from the result of `ac -p` into the students map.

#### Parameters

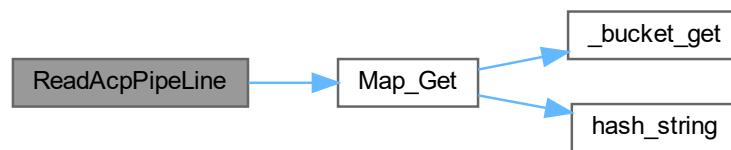
<i>stmap</i>	A map of students.
<i>acp_line</i>	A string representing 1 line result from <code>ac -p</code> .

#### Returns

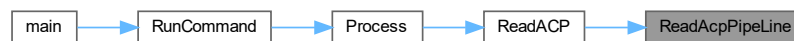
-1 if `acp_line` is NULL or length is less than 1, otherwise 0.

Definition at line 186 of file [Build.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.3.2.7 ReadCumulativeFileLine()

```

void ReadCumulativeFileLine (
    map * time_map,
    char * acp_line )
  
```

Reads a single line from the initial cumulative file and updates the map so that userID maps to a float value in the initial file.

#### Note

A line is structured like this: `mes08346 10.06` It finishes with a line starting with `total`; this line should be disregarded.

#### Parameters

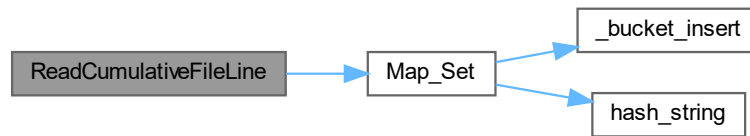
<i>time_map</i>	The cumulative map.
<i>acp_line</i>	A single line from <code>ac -p</code> .

#### Returns

-1 ...

Definition at line 176 of file [Build.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.3.2.8 ReadInitialCumulative()

```

int ReadInitialCumulative (
    map * time_map,
    char * filename )
  
```

Populates the cumulative map by reading from the initial cumulative file. The map will be of the form [userID] -> minutes\_float

The map will contain users who we don't care about, but it doesn't matter.

#### Parameters

<i>time_map</i>	A map of cumulative times. Different from the students map.
<i>filename</i>	The filename where the initial cumulative times are located.

#### Returns

0 if success. -1 if it failed to find the file.

Definition at line 135 of file [Build.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.3.2.9 SetAllStudentsInactive()

```

void SetAllStudentsInactive (
    Student * stud_arr,
    int arr_len )
  
```

Sets the 'active' property on all students in the students array to 0.

#### Parameters

<i>stud_arr</i>	The students array.
<i>arr_len</i>	The length of the students array.

Definition at line 108 of file [Build.c](#).

Here is the caller graph for this function:



### 6.3.2.10 UpdateFromWho()

```
int UpdateFromWho (
    map * stmap )
```

Executes the 'who' command by reading from a file pipe. Calls ProcessWhoLine for each line, to realize updates in the user data from the who command.

#### Parameters

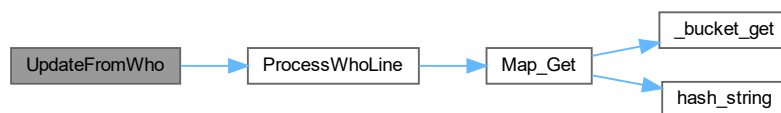
<i>stmap</i>	The student map.
--------------	------------------

#### Returns

0 if succesful, otherwise nonzero.

Definition at line 40 of file [Build.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.3.2.11 WriteStudentsToMemory()

```
void WriteStudentsToMemory (
    void * mem_ptr,
    Student * stud_arr,
    int arr_len )
```

Writes the students array to the location specified by `mem_ptr` (eg. the shared memory segment).

#### Parameters

<i>mem_ptr</i>	The address to write at.
<i>stud_arr</i>	The students array to write.
<i>arr_len</i>	The length of the students array.

Definition at line 117 of file [Build.c](#).

Here is the caller graph for this function:



### 6.3.3 Variable Documentation

#### 6.3.3.1 dirty

```
short dirty [extern]
```

Set to '1' if there are changes that should be written to a file.

Definition at line 38 of file [Build.c](#).

#### 6.3.3.2 students

```
Student* students [extern]
```

A pointer to the students array. It is heap allocated with malloc, when PopulateStudents is called.

#### Note

Generally this array and its length are passed around via parameters, to decouple as much as possible and enable simple testing and dummy data.

Definition at line 14 of file [Build.c](#).

## 6.4 Build.h

[Go to the documentation of this file.](#)

```

00001 #ifndef BUILD_H
00002 #define BUILD_H
00008 #include "Data.h"
00009 #include "map.h"
00010
00011 // ~~~~~ Data Structures ~~~~~
00012
00018 extern Student *students;
00019
00027 void PopulateStudents(char **studentIDs, char **studentNames, int arsize);
00028
00035 void BuildStudentMap(map *stmap, Student *studentArr, int studentArrLength);
00036
00037 // ~~~~~ Processing ~~~~~
00038
00040 extern short dirty;
00041
00048 int UpdateFromWho(map *stmap);
00049
00063 int ProcessWhoLine(map *stmap, char *whoLine, int whoLineLength);
00064
00071 void SetAllStudentsInactive(Student *stud_arr, int arr_len);
00072
00081 void WriteStudentsToMemory(void *mem_ptr, Student *stud_arr, int arr_len);
00082
00083 // ~~~~~ Cumulative Processing ~~~~~
00084
00094 int ReadInitialCumulative(map *time_map, char *filename);
00095
00104 int ReadACP(map *st_map);
00105
00115 void ReadCumulativeFileLine(map *time_map, char *acp_line);
00116
00124 int ReadAcpPipeLine(map *stmap, char *acp_line);
00125
00135 void CalculateCumulative(Student *stud_arr, int stud_arr_len, map *time_map);
00136
00137 #endif

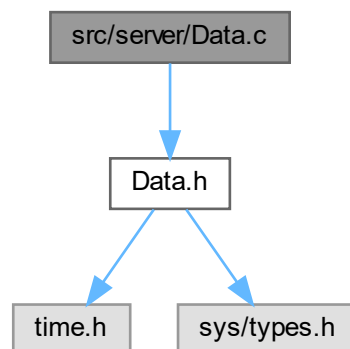
```

## 6.5 src/server/Data.c File Reference

Data structures and constants.

```
#include "Data.h"
```

Include dependency graph for Data.c:





## Variables

- char \* [Data\\_IDs](#) [[DATA\\_NUM\\_RECORDS](#)]
- char \* [Data\\_Names](#) [[DATA\\_NUM\\_RECORDS](#)]

### 6.5.1 Detailed Description

Data structures and constants.

Definition in file [Data.c](#).

### 6.5.2 Variable Documentation

#### 6.5.2.1 Data\_IDs

```
char* Data_IDs[DATA\_NUM\_RECORDS]
```

**Initial value:**

```
= {  
    "chen",  
    "bea1389",  
    "bol4559",  
    "cal6258",  
    "kre5277",  
    "lon1150",  
    "mas9309",  
    "mes08346",  
    "mil7233",  
    "nef9476",  
    "nov7488",  
    "pan9725",  
    "rac3146",  
    "rub4133",  
    "shr5683",  
    "vay3083",  
    "yos2327"}  
}
```

Definition at line 7 of file [Data.c](#).

#### 6.5.2.2 Data\_Names

```
char* Data_Names[DATA\_NUM\_RECORDS]
```

**Initial value:**

```
= {  
    "Weifeng Chen",  
    "Christian Beatty",  
    "Emily Bolles",  
    "Cameron Calhoun",  
    "Ty Kress",  
    "Cody Long",  
    "Caleb Massey",  
    "Christian Messmer",  
    "Karl Miller",  
    "Jeremiah Neff",  
    "Kaitlyn Novacek",  
    "Joshua Panaro",  
    "Caleb Rachocki",  
    "Caleb Ruby",  
    "Paul Shriner",  
    "Alan Vayansky",  
    "Assefa Ayalew Yoseph"}  
}
```

Constant, all user's names.

Definition at line 26 of file [Data.c](#).

## 6.6 Data.c

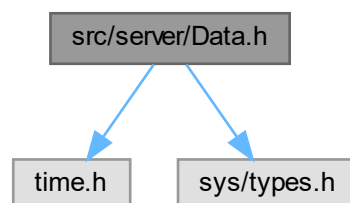
[Go to the documentation of this file.](#)

```
00001
00005 #include "Data.h"
00006
00007 char *Data_IDs[DATA_NUM_RECORDS] = {
00008     "chen",
00009     "beal389",
00010     "bol14559",
00011     "cal6258",
00012     "kre5277",
00013     "lon1150",
00014     "mas9309",
00015     "mes08346",
00016     "mil7233",
00017     "nef9476",
00018     "nov7488",
00019     "pan9725",
00020     "rac3146",
00021     "rub4133",
00022     "shr5683",
00023     "vay3083",
00024     "yos2327"};
00025
00026 char *Data_Names[DATA_NUM_RECORDS] = {
00027     "Weifeng Chen",
00028     "Christian Beatty",
00029     "Emily Bolles",
00030     "Cameron Calhoun",
00031     "Ty Kress",
00032     "Cody Long",
00033     "Caleb Massey",
00034     "Christian Messmer",
00035     "Karl Miller",
00036     "Jeremiah Neff",
00037     "Kaitlyn Novacek",
00038     "Joshua Panaro",
00039     "Caleb Rachocki",
00040     "Caleb Ruby",
00041     "Paul Shriner",
00042     "Alan Vayansky",
00043     "Assefa Ayalew Yoseph"};
```

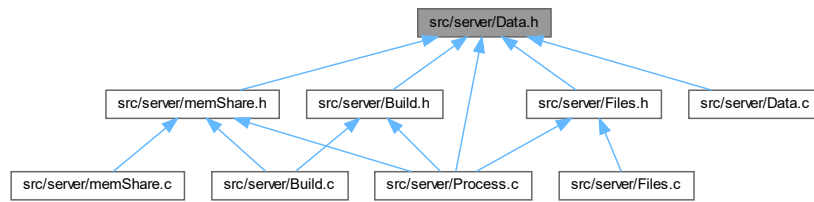
## 6.7 src/server/Data.h File Reference

Declarations of types and macros.

```
#include <time.h>
#include <sys/types.h>
Include dependency graph for Data.h:
```



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



## Data Structures

- struct [Student](#)  
The student data type.

## Macros

- #define [DATA\\_NUM\\_RECORDS](#) 17
- #define [DATA\\_ID\\_MAX\\_LENGTH](#) 9
- #define [DATA\\_NAME\\_MAX\\_LENGTH](#) 21
- #define [DATA\\_SIZE](#) 56

## Variables

- char \* [Data\\_IDs](#) []
- char \* [Data\\_Names](#) []

### 6.7.1 Detailed Description

Declarations of types and macros.

- [DATA\\_NUM\\_RECORDS](#) number of students to have records for
- [DATA\\_ID\\_MAX\\_LENGTH](#) maximum length of [Data\\_IDs](#) can have
- [DATA\\_NAME\\_MAX\\_LENGTH](#) maximum length among [Data\\_Names](#)
- [DATA\\_SIZE](#) total size of a student record

Definition in file [Data.h](#).

### 6.7.2 Macro Definition Documentation

#### 6.7.2.1 DATA\_ID\_MAX\_LENGTH

```
#define DATA_ID_MAX_LENGTH 9
```

The amount of memory (bytes) required to be allocated for the ID field. Equal to the longest name in Data\_IDs, "mes08346", plus the null terminator

Definition at line 24 of file [Data.h](#).

#### 6.7.2.2 DATA\_NAME\_MAX\_LENGTH

```
#define DATA_NAME_MAX_LENGTH 21
```

The amount of memory (bytes) required to be allocated for the Name field. Equal to the longest name in Data\_↔Names, "Assefa Ayalew Yoseph", plus the null terminator

Definition at line 29 of file [Data.h](#).

#### 6.7.2.3 DATA\_NUM\_RECORDS

```
#define DATA_NUM_RECORDS 17
```

The total count of records.

Definition at line 19 of file [Data.h](#).

#### 6.7.2.4 DATA\_SIZE

```
#define DATA_SIZE 56
```

The size of one student record; the result of sizeof(Student).

Definition at line 40 of file [Data.h](#).

### 6.7.3 Variable Documentation

#### 6.7.3.1 Data\_IDs

```
char* Data_IDs[] [extern]
```

Definition at line 7 of file [Data.c](#).

### 6.7.3.2 Data\_Names

```
char* Data_Names[ ] [extern]
```

Constant, all user's names.

Definition at line 26 of file [Data.c](#).

## 6.8 Data.h

[Go to the documentation of this file.](#)

```
00001 #ifndef Data_h
00002 #define Data_h
00013 #include <time.h>
00014 #include <sys/types.h>
00015
00019 #define DATA_NUM_RECORDS 17
00024 #define DATA_ID_MAX_LENGTH 9
00029 #define DATA_NAME_MAX_LENGTH 21
00030
00031 /* Constant, all user IDs. */
00032 extern char *Data_IDs[];
00033
00035 extern char *Data_Names[];
00036
00040 #define DATA_SIZE 56
00041
00045 typedef struct
00046 {
00048     char userID[DATA_ID_MAX_LENGTH];
00050     char fullName[DATA_NAME_MAX_LENGTH];
00052     short age;
00054     float gpa;
00056     short active;
00058     time_t lastLogin;
00060     int loginDuration;
00061 } Student;
00062
00063 #endif
```

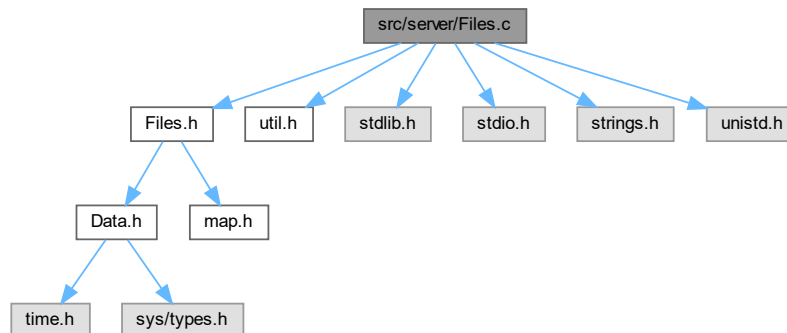
## 6.9 src/server/Files.c File Reference

Declarations of functions that operate on files..

```
#include "Files.h"
#include "util.h"
#include <stdlib.h>
#include <stdio.h>
#include <strings.h>
```

```
#include <unistd.h>
```

Include dependency graph for Files.c:



## Functions

- short [FileExists](#) (char \*file\_name\_to\_check)  
*Determines whether a file exists.*
- int [CreateInitialUserDataFile](#) (char \*file\_name, char \*\*id\_list, int id\_list\_len)  
*Creates the initial user data file. This should be called only the first time the program runs, if it doesn't exist.*
- int [FillStudentMapFromFile](#) (map \*student\_map, char \*file\_name, char \*\*id\_list, int id\_list\_len)  
*Fills the student map with data from the file. It gets age, gpa, and lastLogin from this file.*
- int [WriteStudentArrayToFile](#) (Student \*students, int arr\_len, char \*file\_name)  
*Writes the student array to the file.*
- int [CreateInitialCumulativeFile](#) (char \*file\_name)
- short [DoesLockfileExist](#) ()
- int [CreateLockfile](#) ()
- int [DeleteLockfile](#) ()

### 6.9.1 Detailed Description

Declarations of functions that operate on files..

Definition in file [Files.c](#).

### 6.9.2 Function Documentation

#### 6.9.2.1 CreateInitialCumulativeFile()

```
int CreateInitialCumulativeFile (
    char * file_name )
```

Creates the initial cumulative login time file.

It will hold the result of running 'ac -p'.

**Parameters**

<i>file_name</i>	The name of the file to created. EG STATIC_USER_CUMULATIVE_FILE
------------------	---

**Warning**

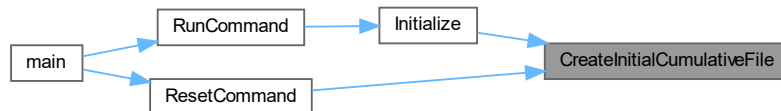
This file should already be validated to not exist.

**Returns**

0 if succesful, -1 if the file couldn't be opened, -2 if the pipe couldn't be opened, otherwise an error code.

Definition at line 96 of file [Files.c](#).

Here is the caller graph for this function:

**6.9.2.2 CreateInitialUserDataFile()**

```

int CreateInitialUserDataFile (
    char * file_name,
    char ** id_list,
    int id_list_len )

```

Creates the initial user data file. This should be called only the first time the program runs, if it doesn't exist.

**Parameters**

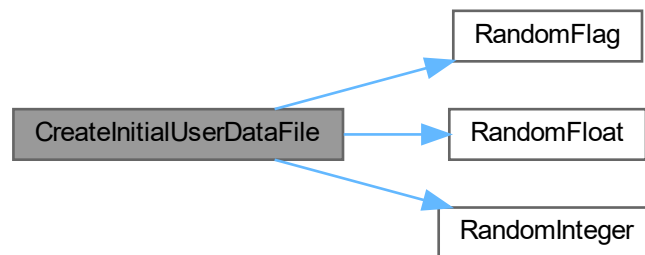
<i>file_name</i>	The file name to create.
<i>id_list</i>	An array containing the IDs. Eg. "Data_IDs" from <a href="#">Data.h</a>
<i>id_list_len</i>	The length of the id_list. Eg. "DATA_NUM_RECORDS" from <a href="#">Data.h</a>

**Returns**

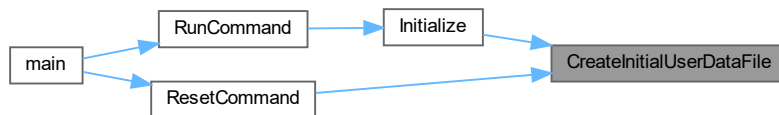
A 0 if the operation was succesful, otherwise nonzero.

Definition at line 27 of file [Files.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.9.2.3 CreateLockfile()

```
int CreateLockfile ( )
```

Creates a lockfile.

#### Warning

This should only be called by a running server process when a lockfile does not already exist.

The lockfile will carry a 'data reset' signal and a process ID. `CreateLockfile` will write the current processes PID.

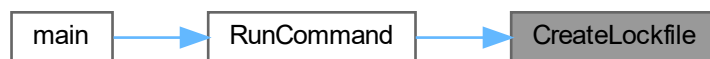


**Returns**

-1 if fopen failed, otherwise 0.

Definition at line 127 of file [Files.c](#).

Here is the caller graph for this function:

**6.9.2.4 DeleteLockfile()**

```
int DeleteLockfile ( )
```

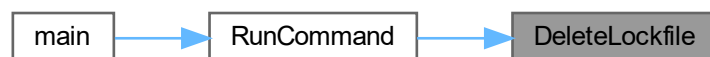
Deletes the lockfile.

**Returns**

0 on success, -1 on failure.

Definition at line 139 of file [Files.c](#).

Here is the caller graph for this function:



### 6.9.2.5 DoesLockfileExist()

```
short DoesLockfileExist ( )
```

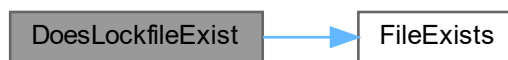
Determines if lockfile exists, which indicates that a server process is already running.

#### Returns

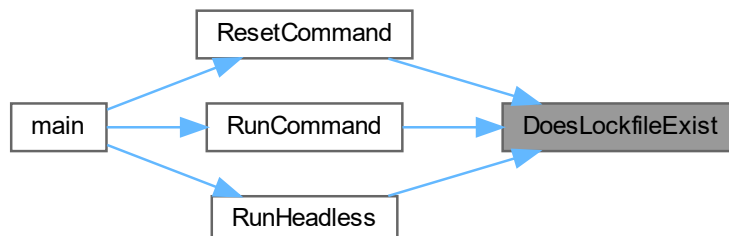
0 if lockfile does not exist, 1 if it does.

Definition at line 122 of file [Files.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.9.2.6 FileExists()

```
short FileExists (
    char * file_name_to_check )
```

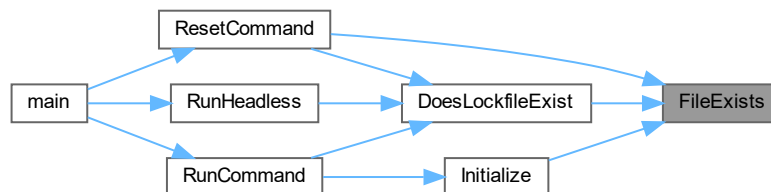
Determines whether a file exists.

**Returns**

1 if it exists. 0 if it does not.

Definition at line 12 of file [Files.c](#).

Here is the caller graph for this function:

**6.9.2.7 FillStudentMapFromFile()**

```

int FillStudentMapFromFile (
    map * student_map,
    char * file_name,
    char ** id_list,
    int id_list_len )

```

Fills the student map with data from the file. It gets age, gpa, and lastLogin from this file.

**Parameters**

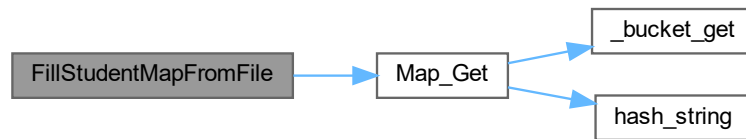
<i>student_map</i>	The map of student structs to be populated from the login.txt file
<i>file_name</i>	The name of the login.txt file.
<i>id_list</i>	An array containing the IDs. Eg. "Data_IDs" from <a href="#">Data.h</a>
<i>id_list_len</i>	The length of the id_list. Eg. "DATA_NUM_RECORDS" from <a href="#">Data.h</a>

**Returns**

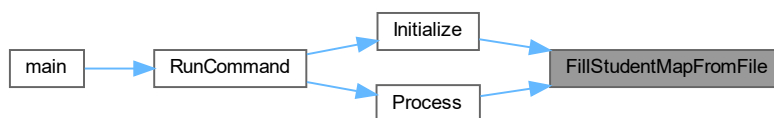
0 if succesful, 1 if there was an error.

Definition at line 53 of file [Files.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.9.2.8 WriteStudentArrayToFile()

```

int WriteStudentArrayToFile (
    Student * students,
    int arr_len,
    char * file_name )
  
```

Writes the student array to the file.

#### Parameters

<i>students</i>	A pointer to the student array that will be read into the file.
<i>arr_len</i>	The length of the students array. e.g. DATA_NUM_RECORDS from <a href="#">Data.h</a> .
<i>file_name</i>	The file name to write.

#### Returns

A 0 if the operation was succesful, otherwise a nonzero.

Definition at line 80 of file [Files.c](#).

Here is the caller graph for this function:



## 6.10 Files.c

[Go to the documentation of this file.](#)

```

00001
00005 #include "Files.h"
00006 #include "util.h"
00007 #include <stdlib.h>
00008 #include <stdio.h>
00009 #include <strings.h>
00010 #include <unistd.h>
00011
00012 short FileExists(char *file_name_to_check)
00013 {
00014     FILE *file = fopen(file_name_to_check, "r");
00015     short result = 1;
00016     if (file == NULL)
00017     {
00018         result = 0;
00019     }
00020     else
00021     {
00022         fclose(file);
00023     }
00024     return result;
00025 }
00026
00027 int CreateInitialUserDataFile(char *file_name, char **id_list, int id_list_len)
00028 {
00029     FILE *file = fopen(file_name, "w");
00030     if (file == NULL)
00031     {
00032         return -1;
00033     }
00034     int i;
00035     for (i = 0; i < id_list_len; i++)
00036     {
00037         int rand_age = RandomInteger(18, 22);
00038         float gpa;
00039         if (RandomFlag(0.42))
00040         {
00041             gpa = 4.0; // 42% of the time, make the GPA 4.0
00042         }
00043         else
00044         {
00045             gpa = RandomFloat(2.5, 4.0);
00046         }
00047         fprintf(file, "%s\t%d\t%.2f\t%d\n", id_list[i], rand_age, gpa, 0);
00048     }
00049     fclose(file);
00050     return 0;
00051 }
00052
00053 int FillStudentMapFromFile(map *student_map, char *file_name, char **id_list, int id_list_len)
00054 {
00055     FILE *file = fopen(file_name, "r");
00056     if (file == NULL)
00057     {
00058         return -1;
00059     }
00060     // id buffer
00061     char user_id[9];
00062     int age;
00063     float gpa;
00064     long time;
00065     while (fscanf(file, "%9s\t%d\t%f\t%d", user_id, &age, &gpa, &time) == 4)
00066     {
00067         map_result result = Map_Get(student_map, user_id);

```

```

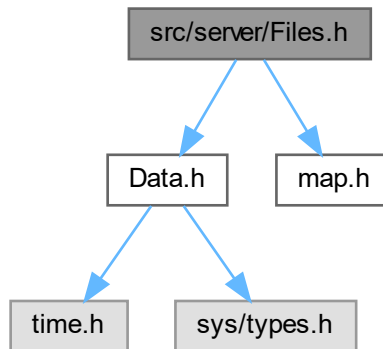
00068         if (result.found == 0)
00069         {
00070             continue;
00071         }
00072         ((Student *)result.data)->age = age;
00073         ((Student *)result.data)->gpa = gpa;
00074         ((Student *)result.data)->lastLogin = time;
00075     }
00076     fclose(file);
00077     return 0;
00078 }
00079
00080 int WriteStudentArrayToFile(Student *students, int arr_len, char *file_name)
00081 {
00082     FILE *file = fopen(file_name, "w");
00083     if (file == NULL)
00084     {
00085         return -1;
00086     }
00087     int i;
00088     for (i = 0; i < arr_len; i++)
00089     {
00090         fprintf(file, "%s\t%d\t%.2f\t%d\n", students[i].userID, students[i].age, students[i].gpa,
students[i].lastLogin);
00091     }
00092     fclose(file);
00093     return 0;
00094 }
00095
00096 int CreateInitialCumulativeFile(char *file_name)
00097 {
00098     FILE *file = fopen(file_name, "w");
00099     if (file == NULL)
00100     {
00101         return -1;
00102     }
00103     FILE *pipe = popen("ac -p", "r");
00104     if (pipe == NULL)
00105     {
00106         fclose(file);
00107         return -2;
00108     }
00109
00110     char line[100];
00111     while (fgets(line, sizeof(line), pipe) != NULL)
00112     {
00113         fputs(line, file);
00114     }
00115     pclose(pipe);
00116     fclose(file);
00117     return 0;
00118 }
00119
00120 // ~~~~~ Lockfile Commands ~~~~~
00121
00122 short DoesLockfileExist()
00123 {
00124     return FileExists(LOCKFILE);
00125 }
00126
00127 int CreateLockfile()
00128 {
00129     FILE *file = fopen(LOCKFILE, "w");
00130     if (file == NULL)
00131     {
00132         return -1;
00133     }
00134     fprintf(file, "0 %d", getpid());
00135     fclose(file);
00136     return 0;
00137 }
00138
00139 int DeleteLockfile()
00140 {
00141     return remove(LOCKFILE);
00142 }

```

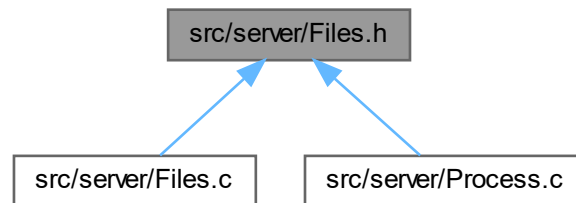
## 6.11 src/server/Files.h File Reference

Definitions for functions that operate on files.

```
#include "Data.h"
#include "map.h"
Include dependency graph for Files.h:
```



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



## Macros

- `#define` `STATIC_USER_DATA_FILE` `"static-user-data.txt"`
- `#define` `STATIC_USER_CUMULATIVE_FILE` `"static-user-cumulative-start.txt"`
- `#define` `LOCKFILE` `"/tmp/ecet-server.lock"`

## Functions

- short `FileExists` (char \*file\_name\_to\_check)  
*Determines whether a file exists.*
- int `CreateInitialUserDataFile` (char \*file\_name, char \*\*id\_list, int id\_list\_len)  
*Creates the initial user data file. This should be called only the first time the program runs, if it doesn't exist.*
- int `WriteStudentArrayToFile` (Student \*students, int arr\_len, char \*file\_name)

*Writes the student array to the file.*

- int [FillStudentMapFromFile](#) (map \*student\_map, char \*file\_name, char \*\*id\_list, int id\_list\_len)  
*Fills the student map with data from the file. It gets age, gpa, and lastLogin from this file.*
- int [CreateInitialCumulativeFile](#) (char \*file\_name)
- short [DoesLockfileExist](#) ()
- int [CreateLockfile](#) ()
- int [DeleteLockfile](#) ()

### 6.11.1 Detailed Description

Definitions for functions that operate on files.

Some program data needs to be stored in files, to preserve it in the case of early termination.

There are three files that are created if they don't exist when the program is first run.

- `STATIC_USER_DATA_FILE` contains a list of userIDs, ages, gpa, and last login time. Age and gpa are randomly generated on server start and when "reset" is run. The login time is updated when it changes as per the dirty flag.
- `STATIC_USER_CUMULATIVE_FILE` contains the results of 'ac -p' run when the server first starts. These values will be subtracted from later pipes of "ac -p" to determine the cumulative time since the server started.
- `LOCKFILE` contains a flag, 0 or 1, that indicates whether the `STATIC_USER_DATA_FILE` has been re-randomized and should be re-read. It contains the process ID of the running server process. It serves as an indicator to the process as to whether a server is already running and, when "close" is passed as a command line argument, which process to kill.

Definition in file [Files.h](#).

### 6.11.2 Macro Definition Documentation

#### 6.11.2.1 LOCKFILE

```
#define LOCKFILE "/tmp/ecet-server.lock"
```

The lockfile serves as a signal to subsequent processes as to whether or not server is already running.

#### Note

File contains the following

- (1) a 1 or a 0 indicating whether the data has been reset and must be re-read
- (2) an integer corresponding to the PID of the process so that server close can end that process

Definition at line 52 of file [Files.h](#).



### 6.11.2.2 STATIC\_USER\_CUMULATIVE\_FILE

```
#define STATIC_USER_CUMULATIVE_FILE "static-user-cumulative-start.txt"
```

File name for the text file that will store the cumulative login time for each user at the point in time when it was created.

The values in this file are subtracted from the result of running 'ac -p' later to get the cumulative time each user was logged in since the server started.

#### Note

Each line contains the following.

- (1) A user ID
- (2) An integer representing the minutes the user has been logged in.

Definition at line 42 of file [Files.h](#).

### 6.11.2.3 STATIC\_USER\_DATA\_FILE

```
#define STATIC_USER_DATA_FILE "static-user-data.txt"
```

File name for the text file that will store user data, namely, the age, gpa, and last login time.

#### Note

Each line contain in the created file contains:

- (1) The ID from the students array, where the `line # - 1 ==` the index of the students array
- (2) A tab character
- (3) A random int between 18 and 22, for the age.
- (4) A tab character
- (5) A random float between 2.5 and 4.0, for the GPA.
- (6) A tab character.
- (7) A 0 (representing the last login time)
- (8) A newline.

The order of entries in the file is the same as the order in the `Data_IDs` array from [Data.c](#).

Definition at line 32 of file [Files.h](#).

## 6.11.3 Function Documentation

### 6.11.3.1 CreateInitialCumulativeFile()

```
int CreateInitialCumulativeFile (  
    char * file_name )
```

Creates the initial cumulative login time file.

It will hold the result of running 'ac -p'.

**Parameters**

<i>file_name</i>	The name of the file to created. EG STATIC_USER_CUMULATIVE_FILE
------------------	---

**Warning**

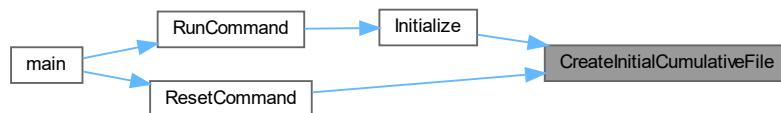
This file should already be validated to not exist.

**Returns**

0 if succesful, -1 if the file couldn't be opened, -2 if the pipe couldn't be opened, otherwise an error code.

Definition at line 96 of file [Files.c](#).

Here is the caller graph for this function:

**6.11.3.2 CreateInitialUserDataFile()**

```

int CreateInitialUserDataFile (
    char * file_name,
    char ** id_list,
    int id_list_len )

```

Creates the initial user data file. This should be called only the first time the program runs, if it doesn't exist.

**Parameters**

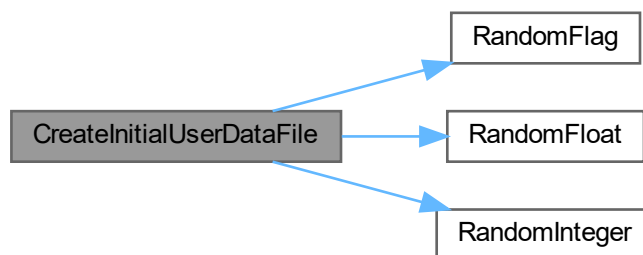
<i>file_name</i>	The file name to create.
<i>id_list</i>	An array containing the IDs. Eg. "Data_IDs" from <a href="#">Data.h</a>
<i>id_list_len</i>	The length of the id_list. Eg. "DATA_NUM_RECORDS" from <a href="#">Data.h</a>

**Returns**

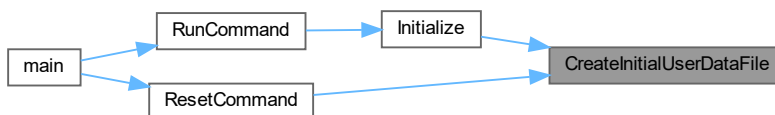
A 0 if the operation was succesful, otherwise nonzero.

Definition at line 27 of file [Files.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.11.3.3 CreateLockfile()

```
int CreateLockfile ( )
```

Creates a lockfile.

#### Warning

This should only be called by a running server process when a lockfile does not already exist.

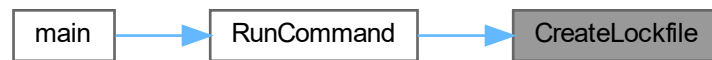
The lockfile will carry a 'data reset' signal and a process ID. `CreateLockfile` will write the current processes PID.

**Returns**

-1 if fopen failed, otherwise 0.

Definition at line 127 of file [Files.c](#).

Here is the caller graph for this function:

**6.11.3.4 DeleteLockfile()**

```
int DeleteLockfile ( )
```

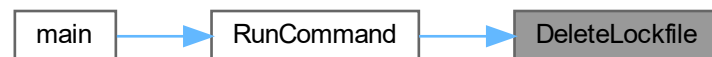
Deletes the lockfile.

**Returns**

0 on success, -1 on failure.

Definition at line 139 of file [Files.c](#).

Here is the caller graph for this function:



### 6.11.3.5 DoesLockfileExist()

```
short DoesLockfileExist ( )
```

Determines if lockfile exists, which indicates that a server process is already running.

#### Returns

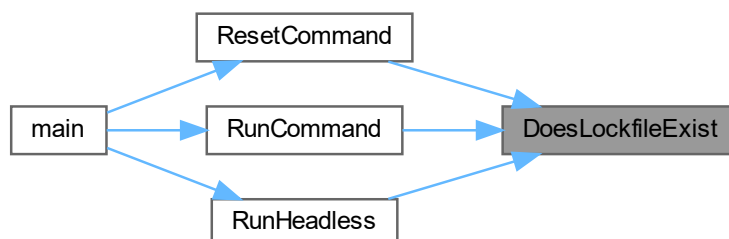
0 if lockfile does not exist, 1 if it does.

Definition at line 122 of file [Files.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.11.3.6 FileExists()

```
short FileExists (
    char * file_name_to_check )
```

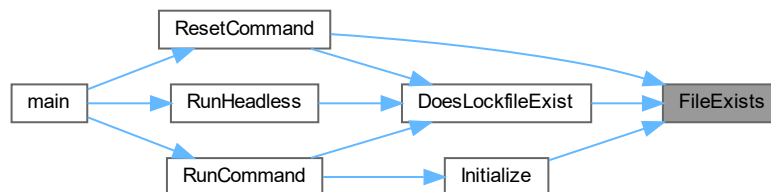
Determines whether a file exists.

**Returns**

1 if it exists. 0 if it does not.

Definition at line 12 of file [Files.c](#).

Here is the caller graph for this function:

**6.11.3.7 FillStudentMapFromFile()**

```

int FillStudentMapFromFile (
    map * student_map,
    char * file_name,
    char ** id_list,
    int id_list_len )

```

Fills the student map with data from the file. It gets age, gpa, and lastLogin from this file.

**Parameters**

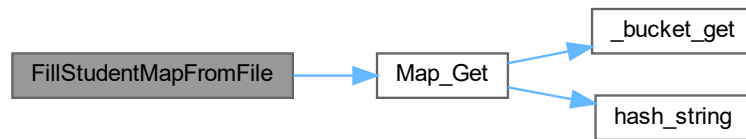
<i>student_map</i>	The map of student structs to be populated from the login.txt file
<i>file_name</i>	The name of the login.txt file.
<i>id_list</i>	An array containing the IDs. Eg. "Data_IDs" from <a href="#">Data.h</a>
<i>id_list_len</i>	The length of the id_list. Eg. "DATA_NUM_RECORDS" from <a href="#">Data.h</a>

**Returns**

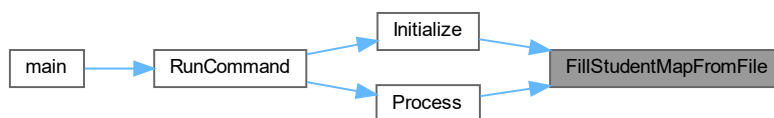
0 if succesful, 1 if there was an error.

Definition at line 53 of file [Files.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.11.3.8 WriteStudentArrayToFile()

```

int WriteStudentArrayToFile (
    Student * students,
    int arr_len,
    char * file_name )
  
```

Writes the student array to the file.

#### Parameters

<i>students</i>	A pointer to the student array that will be read into the file.
<i>arr_len</i>	The length of the students array. e.g. DATA_NUM_RECORDS from <a href="#">Data.h</a> .
<i>file_name</i>	The file name to write.

#### Returns

A 0 if the operation was succesful, otherwise a nonzero.

Definition at line 80 of file [Files.c](#).

Here is the caller graph for this function:



## 6.12 Files.h

[Go to the documentation of this file.](#)

```

00001 #ifndef Files_H
00002 #define Files_H
00014 #include "Data.h"
00015 #include "map.h"
00016
00032 #define STATIC_USER_DATA_FILE "static-user-data.txt"
00033
00042 #define STATIC_USER_CUMULATIVE_FILE "static-user-cumulative-start.txt"
00043
00052 #define LOCKFILE "/tmp/ecet-server.lock"
00053
00058 short FileExists(char *file_name_to_check);
00059
00069 int CreateInitialUserDataFile(char *file_name, char **id_list, int id_list_len);
00070
00080 int WriteStudentArrayToFile(Student *students, int arr_len, char *file_name);
00081
00092 int FillStudentMapFromFile(map *student_map, char *file_name, char **id_list, int id_list_len);
00093
00103 int CreateInitialCumulativeFile(char *file_name);
00104
00105
00106
00107 // ~~~~~ Lockfile Commands ~~~~~
00108
00114 short DoesLockfileExist();
00115
00124 int CreateLockfile();
00125
00130 int DeleteLockfile();
00131
00132 #endif

```

## 6.13 src/server/main.c File Reference

Program entry point.

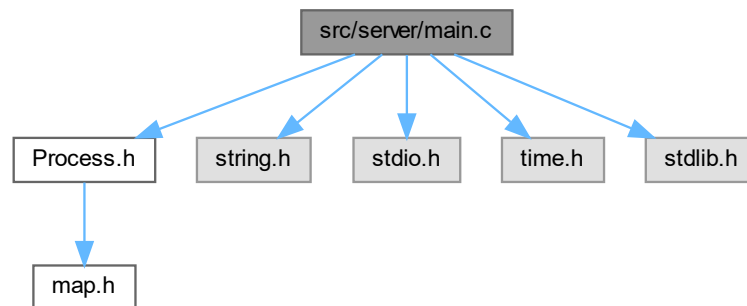
```

#include "Process.h"
#include <string.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

```



Include dependency graph for main.c:



## Functions

- `int main (int argc, char **argv)`  
*Program entry.*

### 6.13.1 Detailed Description

Program entry point.

Definition in file [main.c](#).

### 6.13.2 Function Documentation

#### 6.13.2.1 main()

```
int main (  
    int argc,  
    char ** argv )
```

Program entry.

Parses arguments and calls the appropriate [Process.h](#) function.

Parameters

<code>argc</code>	The argument count.
<code>argv</code>	The argument values.



```

00115     else if (strcmp(argv[1], "run") == 0 || strcmp(argv[1], "start") == 0)
00116     {
00117         RunCommand();
00118     }
00119     else
00120     {
00121         printf("Unknown option!\n");
00122         HelpCommand();
00123     }
00124     return 0;
00125 }

```

## 6.15 src/server/map.c File Reference

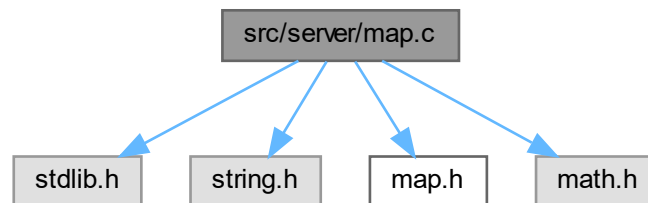
Definitions for functions relating to a hash map data structure.

```

#include "stdlib.h"
#include "string.h"
#include "map.h"
#include "math.h"

```

Include dependency graph for map.c:



### Functions

- int [hash\\_log2](#) (int num\_to\_log)
 

*Get's a log2 ceiling. Eg, `hash_log2(5) == 3`.*
- int [hash\\_upperLimit](#) (int bitsize)
 

*This calculates what the actual capacity of the map will be. Given a result from `hash_log2`, it gets the maximum storable for that many bits. For example, `hash_upperLimit(3)` returns the maximum that 3 bits can hold, which is 8. `hash_upperLimit(4)` returns 16.*
- int [hash\\_string](#) (int hash\_table\_size, char \*string, int strlen)
- [map](#) \* [NewMap](#) (int capacity)
- void [\\_bucket\\_insert](#) (struct \_map\_bucket \*bucket, char \*key, void \*value)
- void [Map\\_Set](#) ([map](#) \*a\_map, char \*key, void \*value)
 

*Sets a value in the map.*
- void [\\_bucket\\_get](#) (struct \_map\_bucket \*bucket, char \*key, [map\\_result](#) \*result)
- [map\\_result](#) [Map\\_Get](#) ([map](#) \*a\_map, char \*key)
 

*Gets a value from the map. It will return a `map_get_result` describing whether it was succesful, and possibly containing the data sought, or NULL if it was unsuccessful.*
- void [\\_bucket\\_delete](#) (struct \_map\_bucket \*bucket, char \*key, short free\_it, [map\\_result](#) \*result)
- [map\\_result](#) [Map\\_Delete](#) ([map](#) \*a\_map, char \*key, short free\_it)
 

*Deletes a key from the map. Returns a `map_get_result` describing whether the delete was succesful and containing the removed data, if extant.*

## Variables

- int `char_ratio` = (int)(sizeof(int) / sizeof(char))

### 6.15.1 Detailed Description

Definitions for functions relating to a hash map data structure.

Definition in file [map.c](#).

### 6.15.2 Function Documentation

#### 6.15.2.1 `_bucket_delete()`

```
void _bucket_delete (
    struct _map_bucket * bucket,
    char * key,
    short free_it,
    map_result * result )
```

Definition at line 124 of file [map.c](#).

Here is the caller graph for this function:

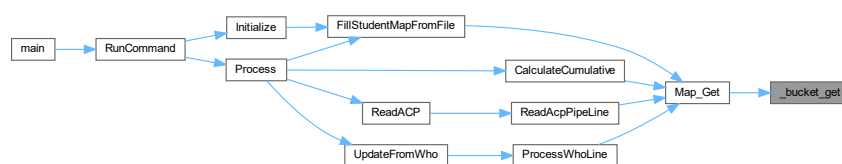


#### 6.15.2.2 `_bucket_get()`

```
void _bucket_get (
    struct _map_bucket * bucket,
    char * key,
    map_result * result )
```

Definition at line 92 of file [map.c](#).

Here is the caller graph for this function:

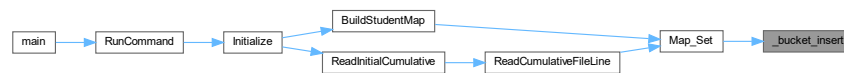


### 6.15.2.3 `_bucket_insert()`

```
void _bucket_insert (
    struct _map_bucket * bucket,
    char * key,
    void * value )
```

Definition at line 64 of file [map.c](#).

Here is the caller graph for this function:



### 6.15.2.4 `hash_log2()`

```
int hash_log2 (
    int number_to_log )
```

Get's a log2 ceiling. Eg, `hash_log2(5) == 3`.

#### Parameters

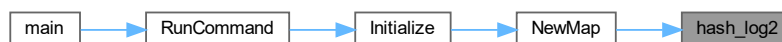
<code>number_to_log</code>	The number to calculate the log of.
----------------------------	-------------------------------------

#### Returns

The log ceiling; eg, the lowest exponent to raise 2 with which would yield a number greater or equal to `number_to_log`.

Definition at line 10 of file [map.c](#).

Here is the caller graph for this function:



### 6.15.2.5 hash\_string()

```
int hash_string (
    int hash_table_capacity,
    char * string,
    int strlen )
```

Uses some clever, prime-number-multiplication, ORing, and bitwise operations to generate a number than, when modulusd with the `hash_table_size`, will produce numbers ('buckets') of even distribution, to minimize the number of collisions. This function contains the meat of the hashing algorithm; it converts a key-string to an array index.

See also

<http://isthe.com/chongo/tech/comp/fnv/>

#### Parameters

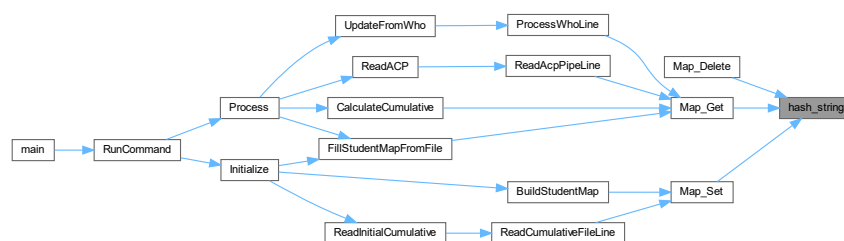
<i>hash_table_capacity</i>	The number of buckets this table holds.
<i>string</i>	The key to hash.
<i>strlen</i>	The length of the key.

#### Returns

The index of the bucket that should be used.

Definition at line 31 of file [map.c](#).

Here is the caller graph for this function:



### 6.15.2.6 hash\_upperLimit()

```
int hash_upperLimit (
    int bitsize )
```

This calculates what the actual capacity of the map will be. Given a result from `hash_log2`, it gets the maximum storable for that many bits. For example, `hash_upperLimit(3)` returns the maximum that 3 bits can hold, which is 8. `hash_upperLimit(4)` returns 16.

## Parameters

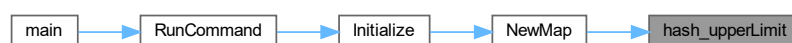
<i>bitsize</i>	The number of bits to calculate the max from.
----------------	---

## Returns

The max value that number of bits can hold.

Definition at line 23 of file [map.c](#).

Here is the caller graph for this function:



### 6.15.2.7 Map\_Delete()

```
map_result Map_Delete (  
    map * a_map,  
    char * key,  
    short free_it )
```

Deletes a key from the map. Returns a `map_get_result` describing whether the delete was succesful and containing the removed data, if extant.

## Parameters

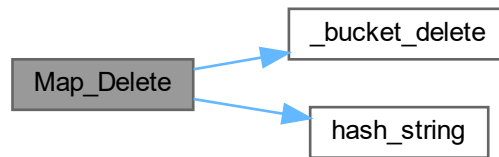
<i>map</i>	The map to delete the key from.
<i>key</i>	The key to delete.
<i>free_it</i>	Whether to call <code>free()</code> on the underlying data.

## Returns

A `map_get_result` with the data that was removed.

Definition at line 150 of file [map.c](#).

Here is the call graph for this function:



### 6.15.2.8 Map\_Get()

```
map_result Map_Get (
    map * a_map,
    char * key )
```

Gets a value from the map. It will return a `map_get_result` describing whether it was succesful, and possibly containing the data sought, or NULL if it was unsuccessful.

#### Parameters

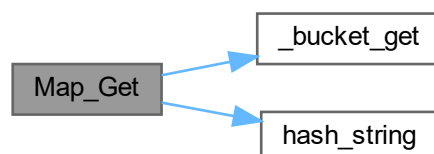
<i>map</i>	The map to retrieve from.
<i>key</i>	The key of the item.

#### Returns

A `map_get_result` containing the sought data.

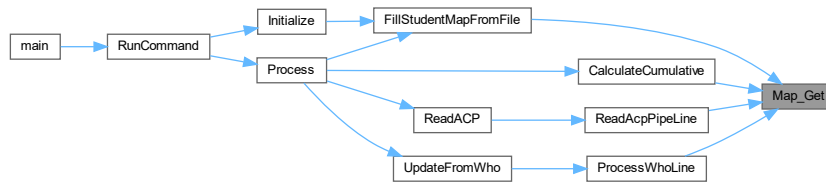
Definition at line 115 of file [map.c](#).

Here is the call graph for this function:





Here is the caller graph for this function:



### 6.15.2.9 Map\_Set()

```

void Map_Set (
    map * a_map,
    char * key,
    void * value )

```

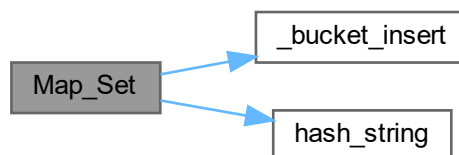
Sets a value in the map.

#### Parameters

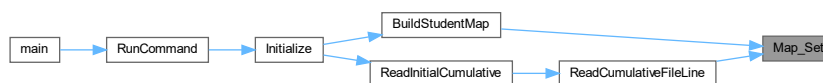
<i>map</i>	The map to set a key in.
<i>key</i>	The key to use.
<i>keylen</i>	The length of the key.
<i>value</i>	The pointer to the data stored at that location.

Definition at line 85 of file `map.c`.

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.15.2.10 NewMap()

```
map * NewMap (
    int capacity )
```

Creates a new map. The map capacity will be a power of 2 that is large enough to contain the estimated size.

#### Parameters

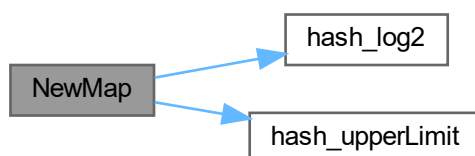
<i>capacity</i>	The estimated required capacity of the map.
-----------------	---

#### Returns

A pointer to the heap allocated map.

Definition at line 46 of file [map.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.15.3 Variable Documentation

#### 6.15.3.1 char\_ratio

```
int char_ratio = (int)(sizeof(int) / sizeof(char))
```

Definition at line 28 of file [map.c](#).

## 6.16 map.c

[Go to the documentation of this file.](#)

```
00001
00005 #include "stdlib.h"
00006 #include "string.h"
00007 #include "map.h"
00008 #include "math.h"
00009
00010 int hash_log2(int num_to_log)
00011 {
00012     int t = 1;
00013     int i = 0;
00014     do
00015     {
00016         num_to_log = num_to_log & ~t;
00017         t = t << 1;
00018         i++;
00019     } while (num_to_log > 0);
00020     return i;
00021 }
00022
00023 int hash_upperLimit(int bitsize)
00024 {
00025     return 1 << bitsize;
00026 }
00027
00028 int char_ratio = (int)(sizeof(int) / sizeof(char));
00029
00030 // Modified some stuff from : http://isthe.com/chongo/tech/comp/fnv/
00031 int hash_string(int hash_table_size, char *string, int strlen)
00032 {
00033     int i, hash = 2166136261;
00034     for (i = 0; i < strlen; i += 1)
00035     {
00036         hash *= 16777619;
00037         hash ^= string[i];
00038     }
00039     if (hash < 0)
00040     {
00041         hash *= -1;
00042     }
00043     return hash % hash_table_size;
00044 }
00045
00046 map *NewMap(int capacity)
00047 {
00048     int log2 = hash_log2(capacity);
00049     int capac = hash_upperLimit(log2);
00050     int sz = sizeof(struct _map_bucket) * capac;
00051     struct _map_bucket *buckets = malloc(sz);
00052     memset(buckets, 0, sz);
00053     int i;
00054     for (i = 0; i < capac; i++)
00055     {
00056         buckets[i] = (struct _map_bucket){NULL, NULL, NULL};
00057     }
00058     map newm = (map){capac, buckets};
00059     map *map_p = malloc(sizeof(map));
00060     *map_p = newm;
00061     return map_p;
00062 }
00063
00064 void _bucket_insert(struct _map_bucket *bucket, char *key, void *value)
```

```

00065 {
00066     struct _map_bucket *check = bucket;
00067     while (check->key != NULL)
00068     {
00069         if (strcmp(check->key, key) == 0)
00070         {
00071             check->data = value;
00072             return;
00073         }
00074         if (check->next == NULL)
00075         {
00076             check->next = malloc(sizeof(struct _map_bucket));
00077             *(check->next) = (struct _map_bucket){NULL, NULL, NULL};
00078         }
00079         check = check->next;
00080     }
00081     check->key = key;
00082     check->data = value;
00083 }
00084
00085 void Map_Set(map *a_map, char *key, void *value)
00086 {
00087     int keyl = (int)strlen(key);
00088     int hash = hash_string(a_map->size, key, keyl);
00089     _bucket_insert(&(a_map->buckets[hash]), key, value);
00090 }
00091
00092 void _bucket_get(struct _map_bucket *bucket, char *key, map_result *result)
00093 {
00094     struct _map_bucket *check = bucket;
00095     while (check->key != NULL)
00096     {
00097         if (strcmp(check->key, key) == 0)
00098         {
00099             result->found = 1;
00100             result->data = check->data;
00101             return;
00102         }
00103         else if (check->next != NULL)
00104         {
00105             check = check->next;
00106         }
00107         else
00108         {
00109             result->found = 0;
00110             break;
00111         }
00112     }
00113 }
00114
00115 map_result Map_Get(map *a_map, char *key)
00116 {
00117     map_result res = (map_result){0, NULL};
00118     int keyl = (int)strlen(key);
00119     int hash = hash_string(a_map->size, key, keyl);
00120     _bucket_get(&(a_map->buckets[hash]), key, &res);
00121     return res;
00122 }
00123
00124 void _bucket_delete(struct _map_bucket *bucket, char *key, short free_it, map_result *result)
00125 {
00126     struct _map_bucket *last = bucket;
00127     struct _map_bucket *next = bucket->next;
00128     while (next != NULL)
00129     {
00130         if (strcmp(next->key, key) == 0)
00131         {
00132             result->found = 1;
00133             result->data = next->data;
00134             if (free_it)
00135             {
00136                 free(next->data);
00137                 result->data = NULL;
00138             }
00139             last->next = next->next;
00140             free(next);
00141         }
00142         else
00143         {
00144             last = next;
00145             next = next->next;
00146         }
00147     }
00148 }
00149
00150 map_result Map_Delete(map *a_map, char *key, short free_it)
00151 {

```

```

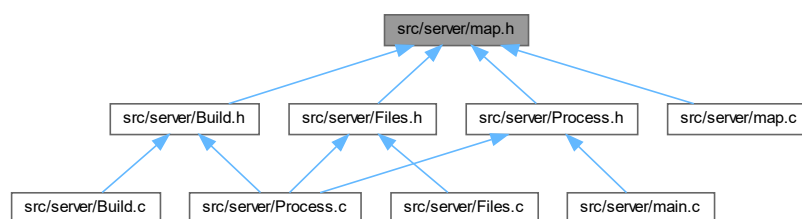
00152     map_result res = (map_result){0, NULL};
00153     int keyl = (int)strlen(key);
00154     int hash = hash_string(a_map->buckets, key, keyl);
00155
00156     struct _map_bucket top = a_map->buckets[hash];
00157     if (top.key == NULL)
00158     {
00159         return res;
00160     }
00161     if (strcmp(top.key, key) == 0)
00162     {
00163         res.found = 1;
00164         res.data = top.data;
00165         if (free_it)
00166         {
00167             free(top.data);
00168             res.data = NULL;
00169         }
00170         if (top.next != NULL)
00171         {
00172             a_map->buckets[hash] = *(top.next);
00173             free(top.next);
00174         }
00175         else
00176         {
00177             a_map->buckets[hash] = (struct _map_bucket){NULL, NULL, NULL};
00178         }
00179         return res;
00180     }
00181     if (top.next == NULL)
00182     {
00183         return res;
00184     }
00185     _bucket_delete(&(a_map->buckets[hash]), key, free_it, &res);
00186
00187     return res;
00188 }

```

## 6.17 src/server/map.h File Reference

Definitions for functions that operate on a hash map data structure.

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



## Data Structures

- struct `map`  
A map. Stores key-value pairs for near constant lookup and insertion time.
- struct `map_result`  
The result of a map retrieval.

## Functions

- `int hash_log2 (int number_to_log)`  
*Get's a log2 ceiling. Eg, `hash_log2(5) == 3`.*
- `int hash_string (int hash_table_capacity, char *string, int strlen)`
- `int hash_upperLimit (int bitsize)`  
*This calculates what the actual capacity of the map will be. Given a result from `hash_log2`, it gets the maximum storable for that many bits. For example, `hash_upperLimit(3)` returns the maximum that 3 bits can hold, which is 8. `hash_upperLimit(4)` returns 16.*
- `map * NewMap (int capacity)`
- `void Map_Set (map *a_map, char *key, void *value)`  
*Sets a value in the map.*
- `map_result Map_Get (map *a_map, char *key)`  
*Gets a value from the map. It will return a `map_get_result` describing whether it was succesful, and possibly containing the data sought, or NULL if it was unsuccessful.*
- `map_result Map_Delete (map *a_map, char *key, short free_it)`  
*Deletes a key from the map. Returns a `map_get_result` describing whether the delete was succesful and containing the removed data, if extant.*

### 6.17.1 Detailed Description

Definitions for functions that operate on a hash map data structure.

Karl's take on a simple hash map structure, which maps strings to void pointers. You can use casting to convert the void pointers into most of whatever else is needed.

Example usage, casting an int into the data part of the map.

```
int myfunc() {
    map *mymap = NewMap(100);
    Map_Set(mymap, "age", (void*)55);
    map_result result = Map_Get(mymap, "age");
    int age;
    if(result.found) {
        age = (int) map_result.data;
    }
}
```

Note, with this simple implementation, the map cannot change its capacity. A change to its capacity would change the hashing.

Ultimately there are really only three things you need to do with the map.

Initialize it, with some capacity larger than you will use. EG `map * mymap = NewMap(100)`. The bigger it is, the fewer collisions (which are pretty rare anyway).

Set some values in it. Eg `Map_Set(mymap, "key", &value)`;

You can cast numbers to void pointers to put them in the map, or you can use the pointers as references to, for example, strings malloced somewhere.

Get some values from it. Eg `void* myval = Map_Get(mymap, "key")`;

Delete some values from it. For example `Map_Delete(mymap, "key", 0)`;

Note that the last parameter, 'free it', tells the map whether it should call 'free' on the underlying data in memory. If this is 1, and the underlying data is not a reference to a malloced part of the heap, errors will result.

Definition in file [map.h](#).

## 6.17.2 Function Documentation

### 6.17.2.1 hash\_log2()

```
int hash_log2 (
    int number_to_log )
```

Get's a log2 ceiling. Eg, hash\_log2(5) == 3.

#### Parameters

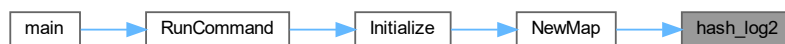
<i>number_to_log</i>	The number to calculate the log of.
----------------------	-------------------------------------

#### Returns

The log ceiling; eg, the lowest exponent to raise 2 with which would yield a number greater or equal to *number\_to\_log*.

Definition at line 10 of file [map.c](#).

Here is the caller graph for this function:



### 6.17.2.2 hash\_string()

```
int hash_string (
    int hash_table_capacity,
    char * string,
    int strlen )
```

Uses some clever, prime-number-multiplication, ORing, and bitwise operations to generate a number than, when modulusd with the *hash\_table\_size*, will produce numbers ('buckets') of even distribution, to minimize the number of collisions. This function contains the meat of the hashing algorithm; it converts a key-string to an array index.

#### See also

<http://isthe.com/chongo/tech/comp/fnv/>

**Parameters**

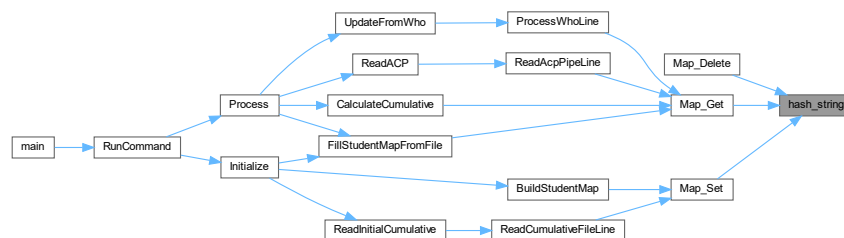
<i>hash_table_capacity</i>	The number of buckets this table holds.
<i>string</i>	The key to hash.
<i>strlen</i>	The length of the key.

**Returns**

The index of the bucket that should be used.

Definition at line 31 of file [map.c](#).

Here is the caller graph for this function:

**6.17.2.3 hash\_upperLimit()**

```
int hash_upperLimit (
    int bitsize )
```

This calculates what the actual capacity of the map will be. Given a result from `hash_log2`, it gets the maximum storable for that many bits. For example, `hash_upperLimit(3)` returns the maximum that 3 bits can hold, which is 8. `hash_upperLimit(4)` returns 16.

**Parameters**

<i>bitsize</i>	The number of bits to calculate the max from.
----------------	---

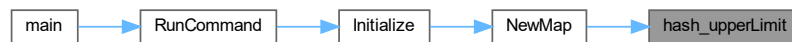
**Returns**

The max value that number of bits can hold.

Definition at line 23 of file [map.c](#).



Here is the caller graph for this function:



#### 6.17.2.4 Map\_Delete()

```

map_result Map_Delete (
    map * a_map,
    char * key,
    short free_it )
  
```

Deletes a key from the map. Returns a `map_get_result` describing whether the delete was succesful and containing the removed data, if extant.

##### Parameters

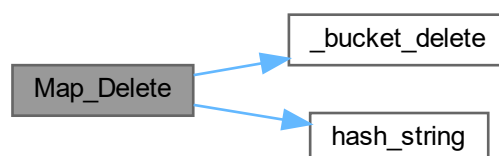
<i>map</i>	The map to delete the key from.
<i>key</i>	The key to delete.
<i>free_it</i>	Whether to call <code>free()</code> on the underlying data.

##### Returns

A `map_get_result` with the data that was removed.

Definition at line 150 of file `map.c`.

Here is the call graph for this function:



### 6.17.2.5 Map\_Get()

```
map_result Map_Get (
    map * a_map,
    char * key )
```

Gets a value from the map. It will return a `map_get_result` describing whether it was succesful, and possibly containing the data sought, or NULL if it was unsuccessful.

#### Parameters

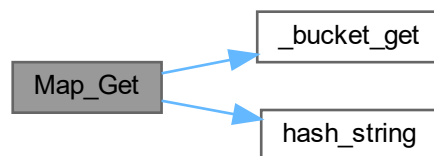
<i>map</i>	The map to retrieve from.
<i>key</i>	The key of the item.

#### Returns

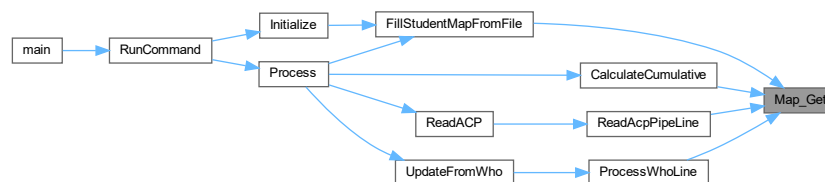
A `map_get_result` containing the sought data.

Definition at line 115 of file [map.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.17.2.6 Map\_Set()

```
void Map_Set (
    map * a_map,
    char * key,
    void * value )
```

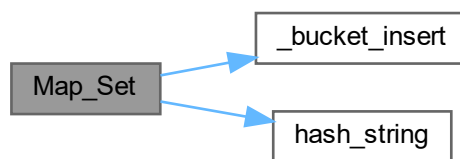
Sets a value in the map.

## Parameters

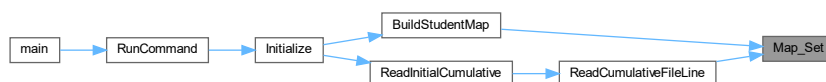
<i>map</i>	The map to set a key in.
<i>key</i>	The key to use.
<i>keylen</i>	The length of the key.
<i>value</i>	The pointer to the data stored at that location.

Definition at line 85 of file [map.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.17.2.7 NewMap()

```
map * NewMap (
    int capacity )
```

Creates a new map. The map capacity will be a power of 2 that is large enough to contain the estimated size.

## Parameters

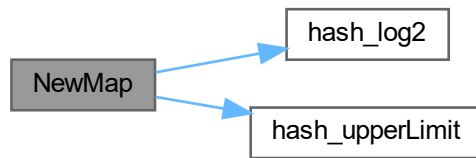
<i>capacity</i>	The estimated required capacity of the map.
-----------------	---

## Returns

A pointer to the heap allocated map.

Definition at line 46 of file [map.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 6.18 map.h

[Go to the documentation of this file.](#)

```

00001 #ifndef map_h
00002 #define map_h
00003
00041 // -----
00042 //           Hashing Math
00043 // -----
00044
00050 int hash_log2(int number_to_log);
00051
00060 int hash_string(int hash_table_capacity, char *string, int strlen);
00061
00067 int hash_upperLimit(int bitsize);
00068
00069 // -----
00070 //           General Map Operations
00071 // -----
00072
00078 struct _map_bucket
00079 {
00080     // The key associated with this bucket.
00081     char *key;
00082     // The data this bucket holds.
00083     void *data;
00084     // The next node in this linked list, or NULL if it is a leaf.
00085     struct _map_bucket *next;
00086 };
00087
00097 typedef struct
00098 {
00099     // The number of base buckets in this map.
00100     int size;
00101     // The buckets for this map.
00102     struct _map_bucket *buckets;
00103 } map;
00104
  
```

```

00108 typedef struct
00109 {
00110     // 1 if succesfully found. 0 if not found.
00111     short found;
00112     // The data linked with that key; indeterminate if found == 0.
00113     void *data;
00114 } map_result;
00115
00122 map *NewMap(int capacity);
00123
00131 void Map_Set(map *a_map, char *key, void *value);
00132
00139 map_result Map_Get(map *a_map, char *key);
00140
00148 map_result Map_Delete(map *a_map, char *key, short free_it);
00149
00150 #endif

```

## 6.19 src/server/memShare.c File Reference

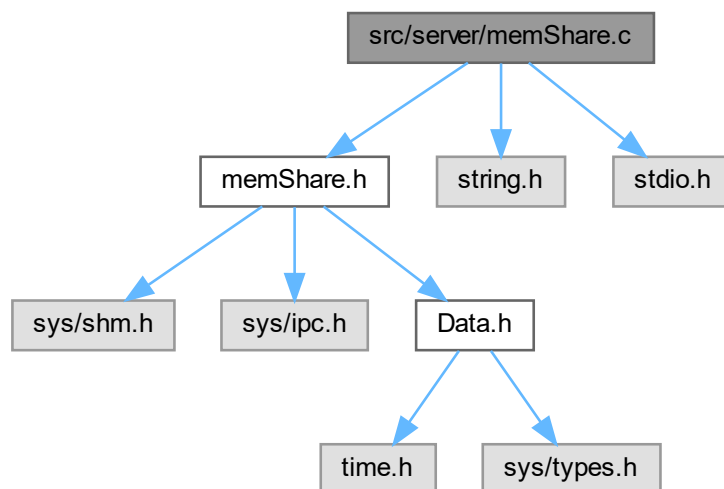
Definitions for functions that operate on a shared memory segment.

```

#include "memShare.h"
#include <string.h>
#include <stdio.h>

```

Include dependency graph for memShare.c:



## Functions

- int [CreateSharedMemory](#) ()
- int [DestroySharedMemory](#) ()
- void \* [GetMemoryPointer](#) (int shared\_mem\_id)
- int [ReleaseMemoryPointer](#) (void \*shmaddr)

### 6.19.1 Detailed Description

Definitions for functions that operate on a shared memory segment.

Definition in file [memShare.c](#).

### 6.19.2 Function Documentation

#### 6.19.2.1 CreateSharedMemory()

```
int CreateSharedMemory ( )
```

CreateSharedMemory retrieves a shared memory ID that can be used to access or delete shared memory.

##### Returns

A shared memory ID that can be used with other 'shm' commands to access shared memory, -1 if an error has occurred

Definition at line 9 of file [memShare.c](#).

Here is the caller graph for this function:



#### 6.19.2.2 DestroySharedMemory()

```
int DestroySharedMemory ( )
```

Flags the shared memory segment for deallocation. Returns the result of that operation.

##### Returns

0 if succesful. 1 if not succesful. Errno will be set.

Definition at line 14 of file [memShare.c](#).

Here is the caller graph for this function:



### 6.19.2.3 GetMemoryPointer()

```
void * GetMemoryPointer (
    int shared_mem_id )
```

"Attaches" to the shared memory, returning a memory pointer to the shared memory.

Calls 'shmat(shared\_mem\_id, NULL, 0)';

#### Parameters

<i>shared_mem_id</i>	The id of the shared memory
----------------------	-----------------------------

#### Returns

A pointer to the shared memory, or -1 if it fails.

Definition at line 23 of file [memShare.c](#).

Here is the caller graph for this function:



### 6.19.2.4 ReleaseMemoryPointer()

```
int ReleaseMemoryPointer (
    void * shmaddr )
```

Release a shm memory pointer.

#### Parameters

<i>shmaddr</i>	The memory pointer to release.
----------------	--------------------------------

#### Returns

Whether the operation was succesful.

Definition at line 28 of file [memShare.c](#).



Here is the caller graph for this function:



## 6.20 memShare.c

[Go to the documentation of this file.](#)

```

00001
00005 #include "memShare.h"
00006 #include <string.h>
00007 #include <stdio.h>
00008
00009 int CreateSharedMemory()
00010 {
00011     return shmget(MEM_KEY, MEM_SIZE, IPC_CREAT | MEM_PERMISSIONS);
00012 }
00013
00014 int DestroySharedMemory()
00015 {
00016     int shm_id = shmget(MEM_KEY, MEM_SIZE, 0);
00017     int control_result = shmctl(shm_id, IPC_RMID, 0);
00018     if (control_result != -1)
00019         return 0;
00020     return control_result;
00021 }
00022
00023 void *GetMemoryPointer(int shared_mem_id)
00024 {
00025     return shmat(shared_mem_id, NULL, 0);
00026 }
00027
00028 int ReleaseMemoryPointer(void *shmaddr)
00029 {
00030     return shmdt(shmaddr);
00031 }
  
```

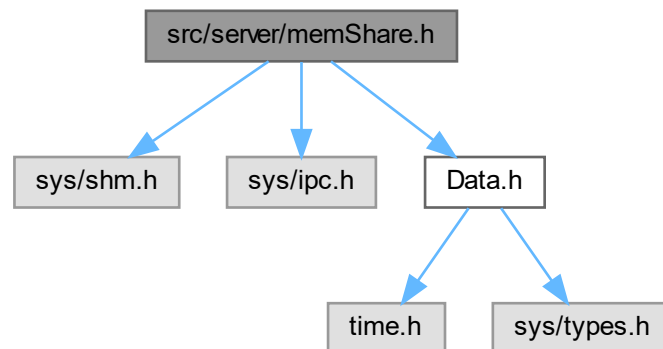
## 6.21 src/server/memShare.h File Reference

Declarations for functions that operate on a shared memory segment.

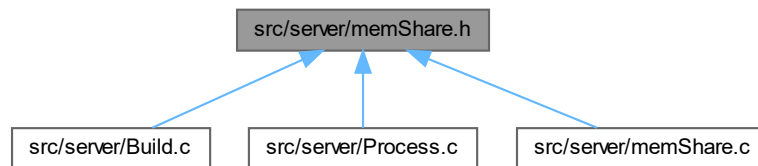
```

#include <sys/shm.h>
#include <sys/ipc.h>
#include "Data.h"
  
```

Include dependency graph for memShare.h:



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



## Macros

- `#define MEM_KEY 0x727`
- `#define MEM_PERMISSIONS 0664`
- `#define MEM_SIZE DATA_SIZE *DATA_NUM_RECORDS`

## Functions

- `int CreateSharedMemory ()`
- `int DestroySharedMemory ()`
- `void * GetMemoryPointer (int shared_mem_id)`
- `int ReleaseMemoryPointer (void *shmaddr)`

### 6.21.1 Detailed Description

Declarations for functions that operate on a shared memory segment.

To share data to clients program uses shared memory

- MEM\_KEY is the key to access the shared memory and clients must have this info
- MEM\_PERMISSIONS who has read, write permissions of the shared memory segment
- MEM\_SIZE the total size of the shared memory allocation

Definition in file [memShare.h](#).

### 6.21.2 Macro Definition Documentation

#### 6.21.2.1 MEM\_KEY

```
#define MEM_KEY 0x727
```

Definition at line 20 of file [memShare.h](#).

#### 6.21.2.2 MEM\_PERMISSIONS

```
#define MEM_PERMISSIONS 0664
```

Definition at line 31 of file [memShare.h](#).

#### 6.21.2.3 MEM\_SIZE

```
#define MEM_SIZE DATA_SIZE *DATA_NUM_RECORDS
```

Definition at line 36 of file [memShare.h](#).

### 6.21.3 Function Documentation

### 6.21.3.1 CreateSharedMemory()

```
int CreateSharedMemory ( )
```

CreateSharedMemory retrieves a shared memory ID that can be used to access or delete shared memory.

#### Returns

A shared memory ID that can be used with other 'shm' commands to access shared memory, -1 if an error has occurred

Definition at line 9 of file [memShare.c](#).

Here is the caller graph for this function:



### 6.21.3.2 DestroySharedMemory()

```
int DestroySharedMemory ( )
```

Flags the shared memory segment for deallocation. Returns the result of that operation.

#### Returns

0 if succesful. 1 if not succesful. Errno will be set.

Definition at line 14 of file [memShare.c](#).

Here is the caller graph for this function:



### 6.21.3.3 GetMemoryPointer()

```
void * GetMemoryPointer (
    int shared_mem_id )
```

"Attaches" to the shared memory, returning a memory pointer to the shared memory.

Calls 'shmat(shared\_mem\_id, NULL, 0)';

## Parameters

<i>shared_mem</i> <i>_id</i>	The id of the shared memory
---------------------------------	-----------------------------

## Returns

A pointer to the shared memory, or -1 if it fails.

Definition at line 23 of file [memShare.c](#).

Here is the caller graph for this function:



### 6.21.3.4 ReleaseMemoryPointer()

```
int ReleaseMemoryPointer (
    void * shmaddr )
```

Release a shm memory pointer.

## Parameters

<i>shmaddr</i>	The memory pointer to release.
----------------	--------------------------------

## Returns

Whether the operation was succesful.

Definition at line 28 of file [memShare.c](#).

Here is the caller graph for this function:



## 6.22 memShare.h

[Go to the documentation of this file.](#)

```

00001 #ifndef MEM_SHARE_H
00002 #define MEM_SHARE_H
00013 #include <sys/shm.h>
00014 #include <sys/ipc.h>
00015 #include "Data.h"
00016
00017 /*
00018     The shared memory key that clients and servers will use to identify the segment.
00019 */
00020 #define MEM_KEY 0x727
00021
00022 /*
00023     Memory permissions are:
00024     Self:   RW    110 = 6
00025     Group:  R     100 = 4
00026     Others: R     100 = 4
00027     - All groups can read.
00028     - Self can write.
00029     - None can execute.
00030 */
00031 #define MEM_PERMISSIONS 0664
00032
00033 /*
00034     The memory allocation must as large as the data size times the number of records.
00035 */
00036 #define MEM_SIZE DATA_SIZE *DATA_NUM_RECORDS
00037
00043 int CreateSharedMemory();
00044
00050 int DestroySharedMemory();
00051
00061 void *GetMemoryPointer(int shared_mem_id);
00062
00068 int ReleaseMemoryPointer(void *shmaddr);
00069
00070 #endif

```

## 6.23 src/server/Process.c File Reference

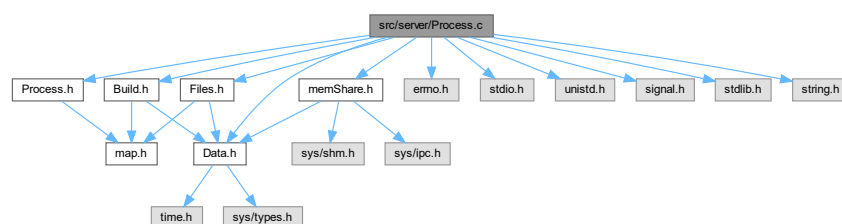
Definitions for functions that manage control flow.

```

#include "Process.h"
#include "Files.h"
#include "Data.h"
#include "Build.h"
#include "memShare.h"
#include <errno.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
#include <string.h>

```

Include dependency graph for Process.c:



## Functions

- int [TerminateExistingServer](#) ()
- int [IndicateRereadNeeded](#) ()
- int [IndicateRereadDone](#) ()
- short [IsRereadNeeded](#) ()
- void [SignalHandle](#) (int signo)
- int [Initialize](#) ()
- void [Process](#) (int shm\_id)
- void [HelpCommand](#) ()
- void [RunCommand](#) ()
- void [StopCommand](#) ()
- void [ResetCommand](#) ()
- void [RunHeadless](#) (char \*processName)

## Variables

- map \* [student\\_map](#)
- map \* [initial\\_cumulative\\_times](#)
- short [is\\_stopping](#) = 0

### 6.23.1 Detailed Description

Definitions for functions that manage control flow.

Definition in file [Process.c](#).

### 6.23.2 Function Documentation

#### 6.23.2.1 HelpCommand()

```
void HelpCommand ( )
```

Displays the commands available to the user.

##### Note

To execute the command, pass "help" as an argument to the program.

This command will also run if arg num is incorrect or if invalid option is entered.

Definition at line [193](#) of file [Process.c](#).

Here is the caller graph for this function:



### 6.23.2.2 IndicateRereadDone()

```
int IndicateRereadDone ( )
```

If we re-read the users file, we can indicate that we have done so by setting the re-read flag back to 0.

#### Warning

should only be called by main process.

#### Returns

0 on success, -1 if the file was not found, otherwise an error number produced by fclose.

Definition at line 56 of file [Process.c](#).

Here is the caller graph for this function:



### 6.23.2.3 IndicateRereadNeeded()

```
int IndicateRereadNeeded ( )
```

If we reset the user data, we need to indicate to the running process that a re-read is needed. This changes the flag in the lockfile to 1, but keeps the same process ID as before there.

#### Warning

should only be called by non main processes

#### Returns

-1 if lockfile not found, 0 if success, or an error number if some other error

Definition at line 39 of file [Process.c](#).

Here is the caller graph for this function:





### 6.23.2.4 Initialize()

```
int Initialize ( )
```

Run once at program start. Calls functions from other modules to do the following:

#### Note

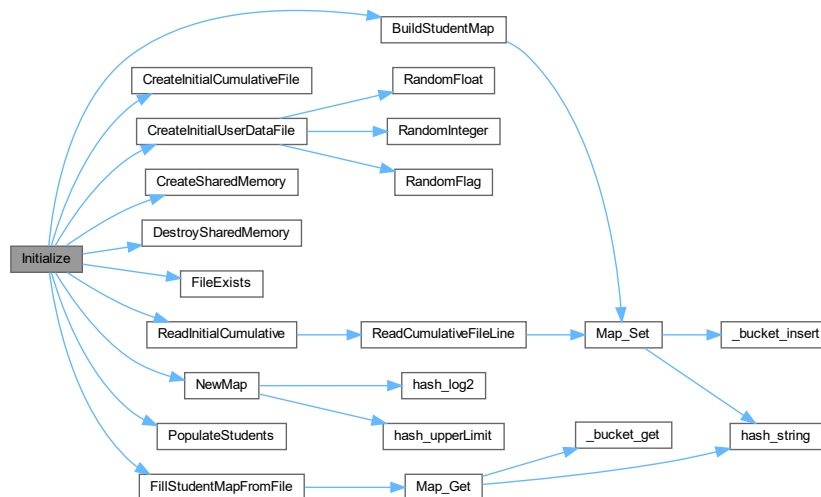
- (1) - Create an initial user data file if it doesn't exist.
- (2) - Initialize the students array.
- (3) - Initialize the students map.
- (4) - Read the data from the user data file into the map/array.
- (5) - Initializes the shared memory segment.

#### Returns

The ID of the shared memory segment or -1 if an error has occurred.

Definition at line 96 of file [Process.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.23.2.5 IsRereadNeeded()

```
short IsRereadNeeded ( )
```

Reads the lockfile for the re-read flag.

#### Warning

Lockfile should exist - should be called by the server in the main process loop

#### Returns

0 if the Lockfile starts with '0', 1 if the Lockfile starts with '1'.

Definition at line 73 of file [Process.c](#).

Here is the caller graph for this function:



### 6.23.2.6 Process()

```
void Process (
    int shm_id )
```

Called repeatedly with a delay.

#### Note

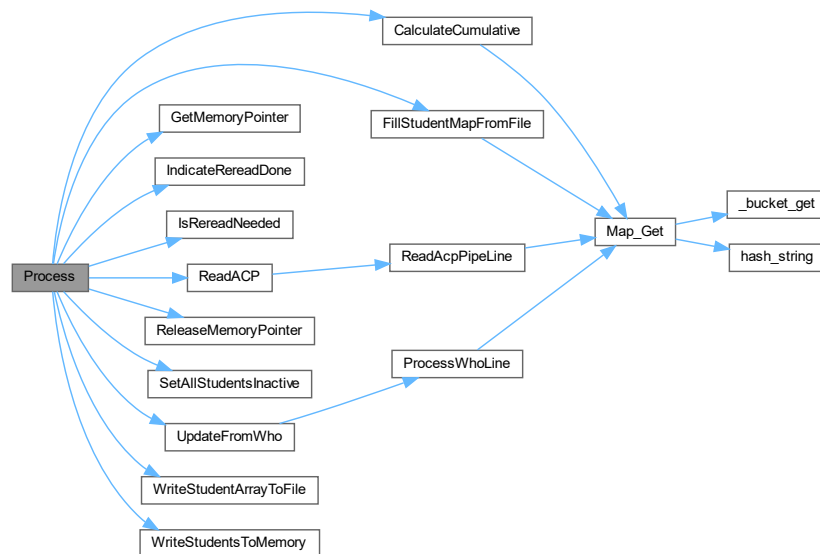
- (1) - Sets all users to inactive.
- (2) - Reads the result of the `who` command, setting some users to active, and possibly changing 'dirty' and last login times.
- (3) - Overwrites the user data file if we are dirty.
- (4) - Sets dirty to false.
- (5) - Rewrites the shared memory.

#### Parameters

<i>shm_id</i>	The ID of the shared memory segment.
---------------	--------------------------------------

Definition at line 146 of file [Process.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.23.2.7 ResetCommand()

```
void ResetCommand ( )
```

Deletes and recreates the static-user-data file and cumulative login file.

#### Note

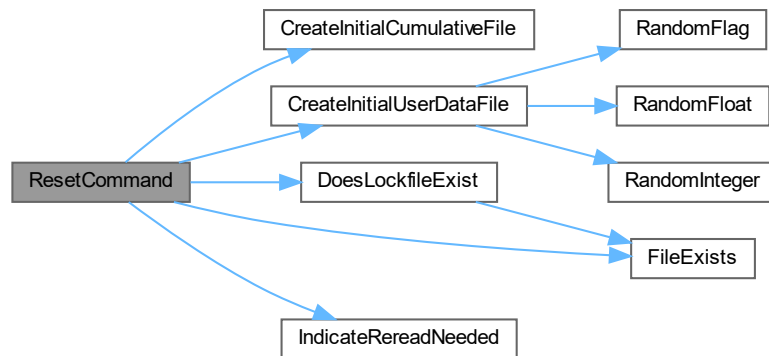
To execute the command, pass "reset" as an argument to the program.

**Warning**

This will clear login times.

Definition at line 267 of file [Process.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**6.23.2.8 RunCommand()**

```
void RunCommand ( )
```

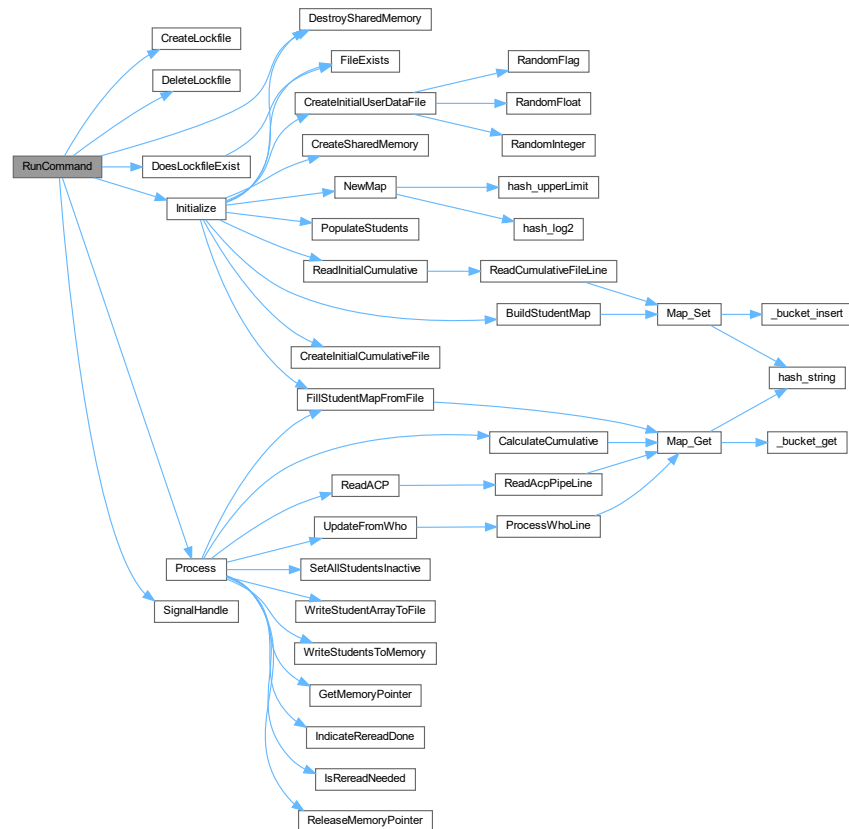
If a server exists, it will be stopped. Then, the process loop will begin.

## Note

To execute the command, pass "run" as an argument to the program.

Definition at line 204 of file [Process.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.23.2.9 RunHeadless()

```
void RunHeadless (
    char * processName )
```

Uses nohup ./{processName} run to run the process headlessly.

**Parameters**

<i>processName</i>	The name of the currently running process.
--------------------	--

Definition at line 306 of file [Process.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**6.23.2.10 SignalHandle()**

```
void SignalHandle (
    int signo )
```

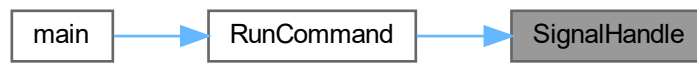
Called by a new server process, telling this server process to shut down. This sets 'is\_stopping' to true, which shuts down the server gracefully, writing any necessary data to the user data file, then deleting the lockfile.

**Parameters**

<i>signo</i>	The signal number will be SIGTERM from the other server process or SIGINT if interrupted from the console.
--------------	--

Definition at line 81 of file [Process.c](#).

Here is the caller graph for this function:



#### 6.23.2.11 StopCommand()

```
void StopCommand ( )
```

Stops an existing server process if it is running.

##### Note

To execute the command, pass "stop" as an argument to the program.

Definition at line [242](#) of file [Process.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.23.2.12 TerminateExistingServer()

```
int TerminateExistingServer ( )
```

Reads the lockfile to get the ID of the process that created it.

Sends a SIGTERM signal to that process.

#### Warning

lockfile should be confirmed to exist

#### Returns

-1 if file doesn't exist, -2 if no valid process ID existed in the file, 1 if sending the kill signal failed.

Definition at line 21 of file [Process.c](#).

Here is the caller graph for this function:



## 6.23.3 Variable Documentation

### 6.23.3.1 initial\_cumulative\_times

```
map* initial_cumulative_times
```

Definition at line 19 of file [Process.c](#).

### 6.23.3.2 is\_stopping

```
short is_stopping = 0
```

If 0, the server is running and looping, re-reading and writing every second. If 1, it is stopping and shutting down.

Definition at line 92 of file [Process.c](#).



## 6.23.3.3 student\_map

```
map* student_map
```

Definition at line 18 of file [Process.c](#).

## 6.24 Process.c

[Go to the documentation of this file.](#)

```
00001
00006 #include "Process.h"
00007 #include "Files.h"
00008 #include "Data.h"
00009 #include "Build.h"
00010 #include "memShare.h"
00011 #include <errno.h>
00012 #include <stdio.h>
00013 #include <unistd.h>
00014 #include <signal.h>
00015 #include <stdlib.h>
00016 #include <string.h>
00017
00018 map *student_map;
00019 map *initial_cumulative_times;
00020
00021 int TerminateExistingServer()
00022 {
00023     FILE *file = fopen(LOCKFILE, "r");
00024     if (file == NULL)
00025     {
00026         return -1;
00027     }
00028     int need_rewrite;
00029     int pid = 0;
00030     fscanf(file, "%d %d", &need_rewrite, &pid);
00031     fclose(file);
00032     if (pid > 0)
00033     {
00034         return kill(pid, SIGTERM);
00035     }
00036     return -2;
00037 }
00038
00039 int IndicateRereadNeeded()
00040 {
00041     FILE *file = fopen(LOCKFILE, "r+");
00042     if (file == NULL)
00043     {
00044         return -1;
00045     }
00046     int err = 0;
00047     err = fseek(file, 0, SEEK_SET);
00048     if (!err)
00049     {
00050         fputc('1', file);
00051     }
00052     err = fclose(file);
00053     return err;
00054 }
00055
00056 int IndicateRereadDone()
00057 {
00058     FILE *file = fopen(LOCKFILE, "r+");
00059     if (file == NULL)
00060     {
00061         return -1;
00062     }
00063     int err = 0;
00064     err = fseek(file, 0, SEEK_SET);
00065     if (!err)
00066     {
00067         fputc('0', file);
00068     }
00069     err = fclose(file);
00070     return err;
00071 }
00072
00073 short IsRereadNeeded()
```

```

00074 {
00075     FILE *file = fopen(LOCKFILE, "r");
00076     char firstc = fgetc(file);
00077     fclose(file);
00078     return firstc == '1';
00079 }
00080
00081 void SignalHandle(int signo)
00082 {
00083     printf("Received shutdown signal.\n");
00084     if (signo == SIGINT || signo == SIGTERM)
00085     {
00086         is_stopping = 1;
00087     }
00088     // possible feature: add a timeout terminate emergency exit (with graceful shutdown)
00089 }
00090
00091
00092 short is_stopping = 0;
00093
00094 // ~~~~~ CLI Commands ~~~~~
00095
00096 int Initialize()
00097 {
00098     int err;
00099     if (!FileExists(STATIC_USER_DATA_FILE))
00100     {
00101         printf("%s does not exist. Creating.\n", STATIC_USER_DATA_FILE);
00102         err = CreateInitialUserDataFile(STATIC_USER_DATA_FILE, Data_IDs, DATA_NUM_RECORDS);
00103         if (err)
00104         {
00105             printf("Problem creating %s!\n", STATIC_USER_DATA_FILE);
00106         }
00107     }
00108     if (!FileExists(STATIC_USER_CUMULATIVE_FILE))
00109     {
00110         printf("%s does not exist. Creating.\n", STATIC_USER_CUMULATIVE_FILE);
00111         err = CreateInitialCumulativeFile(STATIC_USER_CUMULATIVE_FILE);
00112         if (err)
00113         {
00114             printf("Problem creating %s!\n", STATIC_USER_CUMULATIVE_FILE);
00115         }
00116     }
00117     PopulateStudents(Data_IDs, Data_Names, DATA_NUM_RECORDS);
00118     student_map = NewMap(50);
00119     BuildStudentMap(student_map, students, DATA_NUM_RECORDS);
00120     err = FillStudentMapFromFile(student_map, STATIC_USER_DATA_FILE, Data_IDs, DATA_NUM_RECORDS);
00121     if (err)
00122     {
00123         printf("Problem filling student map from %s!\n", STATIC_USER_DATA_FILE);
00124     }
00125     printf("Student data retrieved from file.\n");
00126
00127     initial_cumulative_times = NewMap(50);
00128     err = ReadInitialCumulative(initial_cumulative_times, STATIC_USER_CUMULATIVE_FILE);
00129     if (err)
00130     {
00131         printf("Failed to read %s. Cumulative times may be wrong!", STATIC_USER_CUMULATIVE_FILE);
00132     }
00133
00134     dirty = 0;
00135
00136     int shmid = CreateSharedMemory();
00137     if (shmid == -1)
00138     {
00139         DestroySharedMemory();
00140         shmid = CreateSharedMemory();
00141     }
00142     printf("Shared memory allocated.\n");
00143     return shmid;
00144 }
00145
00146 void Process(int shm_id)
00147 {
00148     if (IsRereadNeeded())
00149     {
00150         printf("\nReread indicated - rechecking user data file.");
00151         FillStudentMapFromFile(student_map, STATIC_USER_DATA_FILE, Data_IDs, DATA_NUM_RECORDS);
00152         IndicateRereadDone();
00153     }
00154     SetAllStudentsInactive(students, DATA_NUM_RECORDS);
00155     int err = ReadACP(student_map);
00156     if (err)
00157     {
00158         printf("Error piping ac -p command! \n");
00159     }
00160     else

```

```

00161     {
00162         CalculateCumulative(students, DATA_NUM_RECORDS, initial_cumulative_times);
00163     }
00164     err = UpdateFromWho(student_map);
00165     if (err)
00166     {
00167         perror("Error updating from who!");
00168     }
00169     if (dirty)
00170     {
00171         err = WriteStudentArrayToFile(students, DATA_NUM_RECORDS, STATIC_USER_DATA_FILE);
00172         if (err)
00173         {
00174             printf("\nError updating %s!", STATIC_USER_DATA_FILE);
00175         }
00176         else
00177         {
00178             dirty = 0;
00179         }
00180     }
00181     void *ptr = GetMemoryPointer(shm_id);
00182     if (ptr == (void *)-1)
00183     {
00184         perror("Error attaching to shared memory");
00185     }
00186     else
00187     {
00188         WriteStudentsToMemory(ptr, students, DATA_NUM_RECORDS);
00189         ReleaseMemoryPointer(ptr);
00190     }
00191 }
00192
00193 void HelpCommand()
00194 {
00195     printf("\nUsage: server [OPTION]\n\n");
00196     printf("Options: \n");
00197     printf("\thelp\t\t\tShows the possible program commands\n");
00198     printf("\treset\t\t\tRegenerates the user data file\n");
00199     printf("\tstop\t\t\tStops an existing server process if it is running\n");
00200     printf("\trun\t\t\tCreates a new server with output to the shell if a server isn't already\n");
00201     printf("\theadless\t\tCreates a new headless server if a server isn't already running.\n\n");
00202 }
00203
00204 void RunCommand()
00205 {
00206     printf("\nRunning server.\n");
00207     if (DoesLockfileExist())
00208     {
00209         printf("\nServer is already running. Run 'server stop' to shut it down first.\n");
00210         return;
00211     }
00212     int err = CreateLockfile();
00213     if (err)
00214     {
00215         printf("\nFailed to create lockfile! Exiting.\n");
00216         return;
00217     }
00218     int shm_id = Initialize();
00219     signal(SIGTERM, SignalHandle);
00220     signal(SIGINT, SignalHandle);
00221     printf("Server started.\n");
00222     fflush(stdout);
00223     while (!is_stopping)
00224     {
00225         Process(shm_id);
00226         sleep(1);
00227     }
00228     printf("Server shutting down.\n");
00229     err = DeleteLockfile();
00230     if (err)
00231     {
00232         printf("Failed to delete lockfile!\n");
00233     }
00234     err = DestroySharedMemory();
00235     if (err)
00236     {
00237         printf("Failed to destroy shared memory!\n");
00238     }
00239     printf("Server terminated.\n");
00240 }
00241
00242 void StopCommand()
00243 {
00244     printf("\nStopping server...\n");
00245     int err = TerminateExistingServer();
00246     if (err)

```

```

00247     {
00248         if (err == -1)
00249         {
00250             printf("Server isn't running.\n");
00251         }
00252         else if (err == -2)
00253         {
00254             printf("Lockfile did not contain a valid process id!\n");
00255         }
00256         else
00257         {
00258             printf("Sending terminate signal failed!\n");
00259         }
00260     }
00261     else
00262     {
00263         printf("Server terminated.\n");
00264     }
00265 }
00266
00267 void ResetCommand()
00268 {
00269     int err;
00270
00271     if (FileExists(STATIC_USER_DATA_FILE))
00272     {
00273         printf("User data file exists. Deleting...\n");
00274         remove(STATIC_USER_DATA_FILE);
00275     }
00276
00277     printf("Creating new data file.\n");
00278     err = CreateInitialUserDataFile(STATIC_USER_DATA_FILE, Data_IDs, DATA_NUM_RECORDS);
00279     if (err)
00280     {
00281         printf("Problem creating %s!\n", STATIC_USER_DATA_FILE);
00282     }
00283     else
00284     {
00285         printf("%s created.\n", STATIC_USER_DATA_FILE);
00286     }
00287
00288     printf("Creating new cumulative file.\n");
00289     err = CreateInitialCumulativeFile(STATIC_USER_CUMULATIVE_FILE);
00290     if (err)
00291     {
00292         printf("Problem creating %s!\n", STATIC_USER_CUMULATIVE_FILE);
00293     }
00294     else
00295     {
00296         printf("%s created.\n", STATIC_USER_CUMULATIVE_FILE);
00297     }
00298
00299     if (DoesLockfileExist())
00300     {
00301         printf("Indicated re-read to running server process.\n");
00302         IndicateRereadNeeded();
00303     }
00304 }
00305
00306 void RunHeadless(char *processName)
00307 {
00308     if (DoesLockfileExist())
00309     {
00310         printf("Server process already running.\n");
00311         return;
00312     }
00313     char commandFront[] = " nohup ";
00314     char commandEnd[] = " run & exit";
00315     size_t comm_length = strlen(commandFront) + strlen(commandEnd) + strlen(processName) + 1;
00316     char *commandFull = malloc(comm_length * sizeof(char));
00317     memset(commandFull, 0, comm_length * sizeof(char));
00318     strcpy(commandFull, commandFront);
00319     strcat(commandFull, processName);
00320     strcat(commandFull, commandEnd);
00321
00322     printf("Executing: %s\n", commandFull);
00323     popen(commandFull, "we");
00324     printf("Server running headlessly.\n");
00325 }

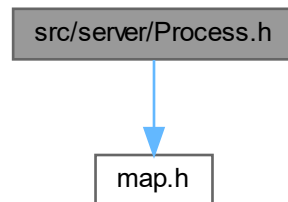
```

## 6.25 src/server/Process.h File Reference

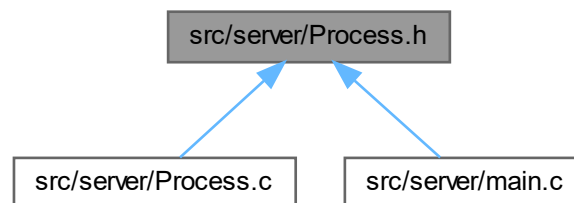
Declarations for functions that manage control flow. This module handles the processes that this server might execute. It calls functions from the other modules to realize program changes.

```
#include "map.h"
```

Include dependency graph for Process.h:



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



### Functions

- int [TerminateExistingServer](#) ()
- int [IndicateRereadNeeded](#) ()
- int [IndicateRereadDone](#) ()
- short [IsRereadNeeded](#) ()
- void [SignalHandle](#) (int signo)
- int [Initialize](#) ()
- void [Process](#) (int shm\_id)
- void [ResetCommand](#) ()
- void [StopCommand](#) ()
- void [RunCommand](#) ()
- void [HelpCommand](#) ()
- void [RunHeadless](#) (char \*processName)

## Variables

- short [is\\_stopping](#)
- [map](#) \* [student\\_map](#)

### 6.25.1 Detailed Description

Declarations for functions that manage control flow. This module handles the processes that this server might execute. It calls functions from the other modules to realize program changes.

Definition in file [Process.h](#).

### 6.25.2 Function Documentation

#### 6.25.2.1 HelpCommand()

```
void HelpCommand ( )
```

Displays the commands available to the user.

##### Note

To execute the command, pass "help" as an argument to the program.

This command will also run if arg num is incorrect or if invalid option is entered.

Definition at line [193](#) of file [Process.c](#).

Here is the caller graph for this function:



### 6.25.2.2 IndicateRereadDone()

```
int IndicateRereadDone ( )
```

If we re-read the users file, we can indicate that we have done so by setting the re-read flag back to 0.

#### Warning

should only be called by main process.

#### Returns

0 on success, -1 if the file was not found, otherwise an error number produced by fclose.

Definition at line 56 of file [Process.c](#).

Here is the caller graph for this function:



### 6.25.2.3 IndicateRereadNeeded()

```
int IndicateRereadNeeded ( )
```

If we reset the user data, we need to indicate to the running process that a re-read is needed. This changes the flag in the lockfile to 1, but keeps the same process ID as before there.

#### Warning

should only be called by non main processes

#### Returns

-1 if lockfile not found, 0 if success, or an error number if some other error

Definition at line 39 of file [Process.c](#).

Here is the caller graph for this function:



### 6.25.2.4 Initialize()

```
int Initialize ( )
```

Run once at program start. Calls functions from other modules to do the following:

#### Note

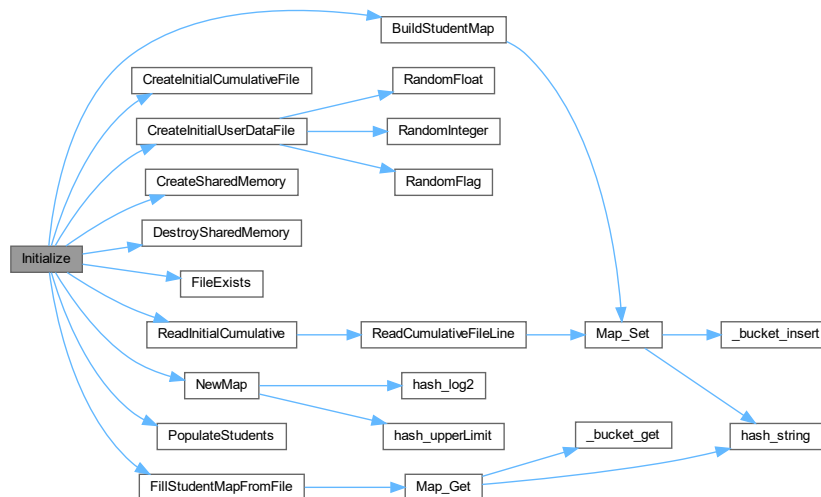
- (1) - Create an initial user data file if it doesn't exist.
- (2) - Initialize the students array.
- (3) - Initialize the students map.
- (4) - Read the data from the user data file into the map/array.
- (5) - Initializes the shared memory segment.

#### Returns

The ID of the shared memory segment or -1 if an error has occurred.

Definition at line 96 of file [Process.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:





### 6.25.2.5 IsRereadNeeded()

```
short IsRereadNeeded ( )
```

Reads the lockfile for the re-read flag.

#### Warning

Lockfile should exist - should be called by the server in the main process loop

#### Returns

0 if the Lockfile starts with '0', 1 if the Lockfile starts with '1'.

Definition at line 73 of file [Process.c](#).

Here is the caller graph for this function:



### 6.25.2.6 Process()

```
void Process (
    int shm_id )
```

Called repeatedly with a delay.

#### Note

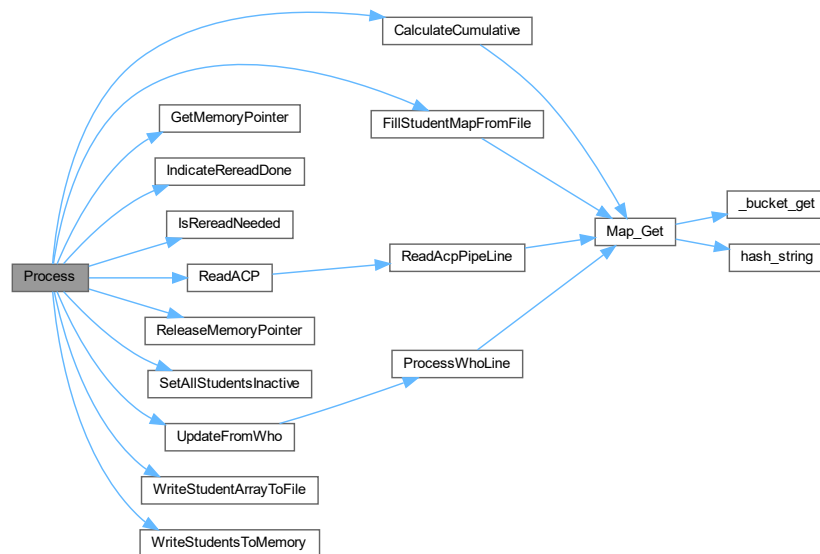
- (1) - Sets all users to inactive.
- (2) - Reads the result of the `who` command, setting some users to active, and possibly changing 'dirty' and last login times.
- (3) - Overwrites the user data file if we are dirty.
- (4) - Sets dirty to false.
- (5) - Rewrites the shared memory.

#### Parameters

<i>shm_id</i>	The ID of the shared memory segment.
---------------	--------------------------------------

Definition at line 146 of file [Process.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.25.2.7 ResetCommand()

```
void ResetCommand ( )
```

Deletes and recreates the static-user-data file and cumulative login file.

#### Note

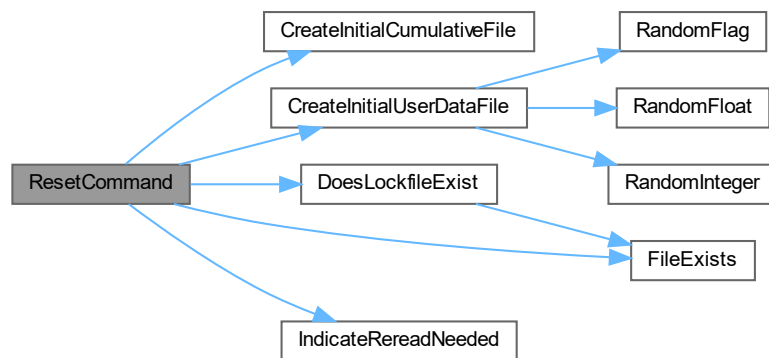
To execute the command, pass "reset" as an argument to the program.

**Warning**

This will clear login times.

Definition at line 267 of file [Process.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**6.25.2.8 RunCommand()**

```
void RunCommand ( )
```

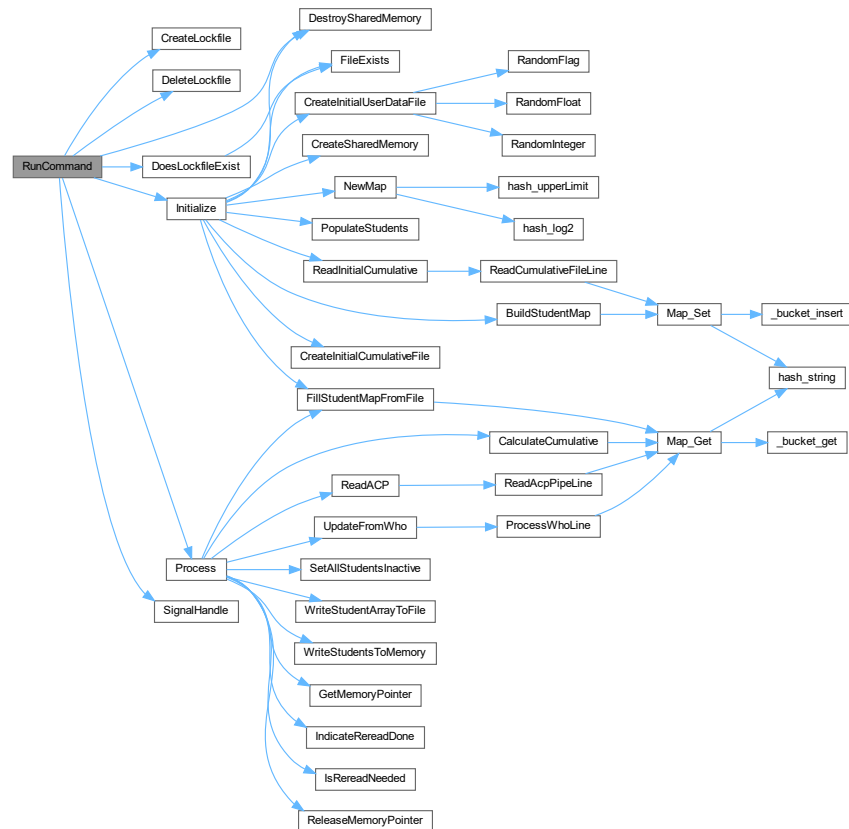
If a server exists, it will be stopped. Then, the process loop will begin.

## Note

To execute the command, pass "run" as an argument to the program.

Definition at line 204 of file [Process.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.25.2.9 RunHeadless()

```
void RunHeadless (
    char * processName )
```

Uses nohup ./{processName} run to run the process headlessly.

**Parameters**

<i>processName</i>	The name of the currently running process.
--------------------	--

Definition at line 306 of file [Process.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**6.25.2.10 SignalHandle()**

```
void SignalHandle (  
    int signo )
```

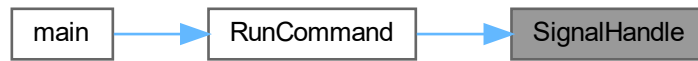
Called by a new server process, telling this server process to shut down. This sets 'is\_stopping' to true, which shuts down the server gracefully, writing any necessary data to the user data file, then deleting the lockfile.

**Parameters**

<i>signo</i>	The signal number will be SIGTERM from the other server process or SIGINT if interrupted from the console.
--------------	--

Definition at line 81 of file [Process.c](#).

Here is the caller graph for this function:



#### 6.25.2.11 StopCommand()

```
void StopCommand ( )
```

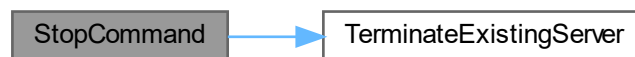
Stops an existing server process if it is running.

##### Note

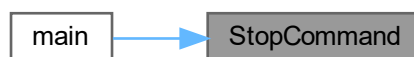
To execute the command, pass "stop" as an argument to the program.

Definition at line [242](#) of file [Process.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.25.2.12 TerminateExistingServer()

```
int TerminateExistingServer ( )
```

Reads the lockfile to get the ID of the process that created it.

Sends a SIGTERM signal to that process.

#### Warning

lockfile should be confirmed to exist

#### Returns

-1 if file doesn't exist, -2 if no valid process ID existed in the file, 1 if sending the kill signal failed.

Definition at line 21 of file [Process.c](#).

Here is the caller graph for this function:



## 6.25.3 Variable Documentation

### 6.25.3.1 is\_stopping

```
short is_stopping [extern]
```

If 0, the server is running and looping, re-reading and writing every second. If 1, it is stopping and shutting down.

Definition at line 92 of file [Process.c](#).

### 6.25.3.2 student\_map

```
map* student_map [extern]
```

Definition at line 18 of file [Process.c](#).

## 6.26 Process.h

[Go to the documentation of this file.](#)

```

00001 #ifndef Process_h
00002 #define Process_h
00008 #include "map.h"
00009
00010
00011
00020 int TerminateExistingServer();
00021
00029 int IndicateRereadNeeded();
00030
00037 int IndicateRereadDone();
00038
00045 short IsRereadNeeded();
00046
00052 void SignalHandle(int signo);
00053
00057 extern short is_stopping;
00058
00059 // ~~~~~ CLI Commands ~~~~~
00060
00061 extern map *student_map;
00062
00073 int Initialize();
00074
00087 void Process(int shm_id);
00088
00095 void ResetCommand();
00096
00102 void StopCommand();
00103
00109 void RunCommand();
00110
00117 void HelpCommand();
00118
00123 void RunHeadless(char *processName);
00124
00125 #endif

```

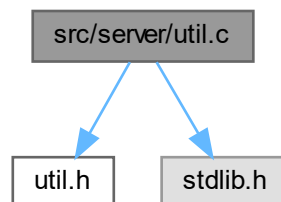
## 6.27 src/server/util.c File Reference

Definitions for helper functions.

```
#include "util.h"
```

```
#include <stdlib.h>
```

Include dependency graph for util.c:



### Functions

- int [RandomInteger](#) (int min, int max)
- float [RandomFloat](#) (float min, float max)
- short [RandomFlag](#) (float percentage\_chance)



### 6.27.1 Detailed Description

Definitions for helper functions.

Definition in file [util.c](#).

### 6.27.2 Function Documentation

#### 6.27.2.1 RandomFlag()

```
short RandomFlag (  
    float percentage_chance )
```

Returns 1, percentage\_chance of the time.

##### Parameters

<i>percentage_chance</i>	The chance to return 1.
--------------------------	-------------------------

##### Note

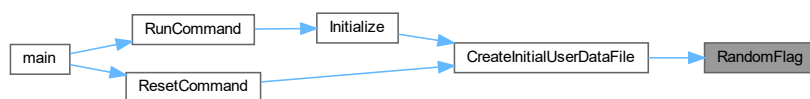
If `percentage_chance > 1`, this will always return true.

##### Returns

1 or 0

Definition at line 22 of file [util.c](#).

Here is the caller graph for this function:



#### 6.27.2.2 RandomFloat()

```
float RandomFloat (  
    float min,  
    float max )
```

Returns a float between min and max.

**Parameters**

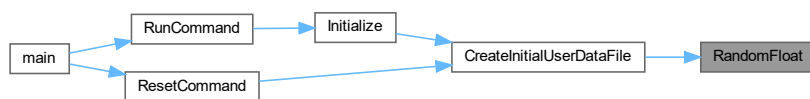
<i>min</i>	The minimum, inclusive.
<i>max</i>	The maximum, inclusive.

**Returns**

A random integer between min and max.

Definition at line 15 of file [util.c](#).

Here is the caller graph for this function:

**6.27.2.3 RandomInteger()**

```
int RandomInteger (
    int min,
    int max )
```

Returns an integer between min and max.

**Parameters**

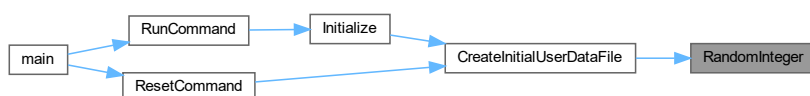
<i>min</i>	The minimum, inclusive.
<i>max</i>	The maximum, inclusive.

**Returns**

A random integer between min and max.

Definition at line 9 of file [util.c](#).

Here is the caller graph for this function:



## 6.28 util.c

[Go to the documentation of this file.](#)

```

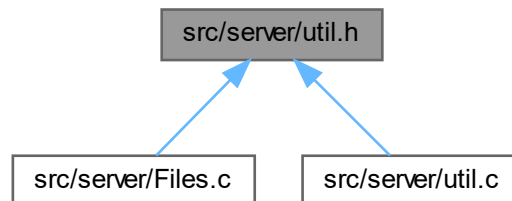
00001
00005 #include "util.h"
00006
00007 #include <stdlib.h>
00008
00009 int RandomInteger(int min, int max)
00010 {
00011     int r_add = rand() % (max - min + 1);
00012     return r_add + min;
00013 }
00014
00015 float RandomFloat(float min, float max)
00016 {
00017     float dif = max - min;
00018     int rand_int = rand() % (int)(dif * 10000);
00019     return min + (float)rand_int / 10000.0;
00020 }
00021
00022 short RandomFlag(float percentage_chance)
00023 {
00024     float random_value = (float)rand() / RAND_MAX;
00025     if (random_value < percentage_chance)
00026     {
00027         return 1;
00028     }
00029     return 0;
00030 }

```

## 6.29 src/server/util.h File Reference

Declarations for helper functions.

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



### Functions

- int [RandomInteger](#) (int min, int max)
- float [RandomFloat](#) (float min, float max)
- short [RandomFlag](#) (float percentage\_chance)

### 6.29.1 Detailed Description

Declarations for helper functions.

Contains utility functions that are not coupled to any other data or structures in the program.

Definition in file [util.h](#).

## 6.29.2 Function Documentation

### 6.29.2.1 RandomFlag()

```
short RandomFlag (
    float percentage_chance )
```

Returns 1, *percentage\_chance* of the time.

#### Parameters

<i>percentage_chance</i>	The chance to return 1.
--------------------------	-------------------------

#### Note

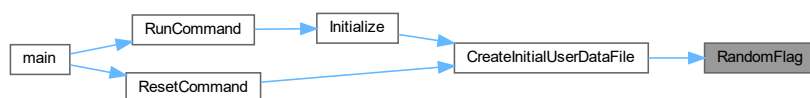
If *percentage\_chance* > 1, this will always return true.

#### Returns

1 or 0

Definition at line 22 of file [util.c](#).

Here is the caller graph for this function:



### 6.29.2.2 RandomFloat()

```
float RandomFloat (
    float min,
    float max )
```

Returns a float between *min* and *max*.

#### Parameters

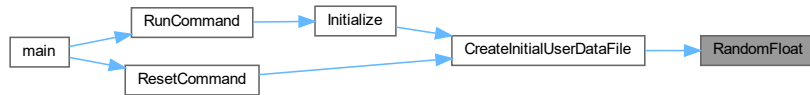
<i>min</i>	The minimum, inclusive.
<i>max</i>	The maximum, inclusive.

**Returns**

A random integer between min and max.

Definition at line 15 of file [util.c](#).

Here is the caller graph for this function:

**6.29.2.3 RandomInteger()**

```

int RandomInteger (
    int min,
    int max )
  
```

Returns an integer between min and max.

**Parameters**

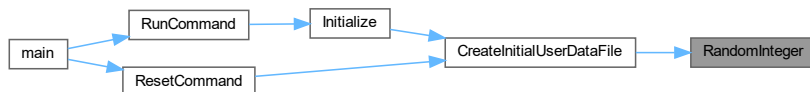
<i>min</i>	The minimum, inclusive.
<i>max</i>	The maximum, inclusive.

**Returns**

A random integer between min and max.

Definition at line 9 of file [util.c](#).

Here is the caller graph for this function:

**6.30 util.h**

[Go to the documentation of this file.](#)

```
00001 #ifndef util_h
00002 #define util_h
00016 int RandomInteger(int min, int max);
00017
00024 float RandomFloat(float min, float max);
00025
00032 short RandomFlag(float percentage_chance);
00033
00034 #endif
```

# Index

- `_bucket_delete`
    - `map.c`, [68](#)
  - `_bucket_get`
    - `map.c`, [68](#)
  - `_bucket_insert`
    - `map.c`, [68](#)
- active
  - Student, [12](#)
- age
  - Student, [12](#)
- buckets
  - `map`, [10](#)
- Build.c
  - BuildStudentMap, [16](#)
  - CalculateCumulative, [17](#)
  - dirty, [25](#)
  - PopulateStudents, [18](#)
  - ProcessWhoLine, [18](#)
  - ReadACP, [19](#)
  - ReadAcpPipeLine, [20](#)
  - ReadCumulativeFileLine, [21](#)
  - ReadInitialCumulative, [22](#)
  - SetAllStudentsInactive, [23](#)
  - students, [25](#)
  - UpdateFromWho, [23](#)
  - WriteStudentsToMemory, [24](#)
- Build.h
  - BuildStudentMap, [30](#)
  - CalculateCumulative, [31](#)
  - dirty, [39](#)
  - PopulateStudents, [31](#)
  - ProcessWhoLine, [32](#)
  - ReadACP, [33](#)
  - ReadAcpPipeLine, [34](#)
  - ReadCumulativeFileLine, [35](#)
  - ReadInitialCumulative, [36](#)
  - SetAllStudentsInactive, [37](#)
  - students, [39](#)
  - UpdateFromWho, [37](#)
  - WriteStudentsToMemory, [38](#)
- BuildStudentMap
  - Build.c, [16](#)
  - Build.h, [30](#)
- CalculateCumulative
  - Build.c, [17](#)
  - Build.h, [31](#)
- char\_ratio
  - `map.c`, [75](#)
- CreateInitialCumulativeFile
  - `Files.c`, [46](#)
  - `Files.h`, [57](#)
- CreateInitialUserDataFile
  - `Files.c`, [47](#)
  - `Files.h`, [58](#)
- CreateLockfile
  - `Files.c`, [48](#)
  - `Files.h`, [59](#)
- CreateSharedMemory
  - `memShare.c`, [87](#)
  - `memShare.h`, [91](#)
- data
  - `map_result`, [10](#)
- Data.c
  - Data\_IDs, [41](#)
  - Data\_Names, [41](#)
- Data.h
  - DATA\_ID\_MAX\_LENGTH, [43](#)
  - Data\_IDs, [44](#)
  - DATA\_NAME\_MAX\_LENGTH, [44](#)
  - Data\_Names, [44](#)
  - DATA\_NUM\_RECORDS, [44](#)
  - DATA\_SIZE, [44](#)
- DATA\_ID\_MAX\_LENGTH
  - Data.h, [43](#)
- Data\_IDs
  - Data.c, [41](#)
  - Data.h, [44](#)
- DATA\_NAME\_MAX\_LENGTH
  - Data.h, [44](#)
- Data\_Names
  - Data.c, [41](#)
  - Data.h, [44](#)
- DATA\_NUM\_RECORDS
  - Data.h, [44](#)
- DATA\_SIZE
  - Data.h, [44](#)
- DeleteLockfile
  - `Files.c`, [49](#)
  - `Files.h`, [60](#)
- DestroySharedMemory
  - `memShare.c`, [87](#)
  - `memShare.h`, [92](#)
- dirty
  - Build.c, [25](#)
  - Build.h, [39](#)
- DoesLockfileExist

- Files.c, [49](#)
- Files.h, [60](#)
- FileExists
  - Files.c, [50](#)
  - Files.h, [61](#)
- Files.c
  - CreateInitialCumulativeFile, [46](#)
  - CreateInitialUserDataFile, [47](#)
  - CreateLockfile, [48](#)
  - DeleteLockfile, [49](#)
  - DoesLockfileExist, [49](#)
  - FileExists, [50](#)
  - FillStudentMapFromFile, [51](#)
  - WriteStudentArrayToFile, [52](#)
- Files.h
  - CreateInitialCumulativeFile, [57](#)
  - CreateInitialUserDataFile, [58](#)
  - CreateLockfile, [59](#)
  - DeleteLockfile, [60](#)
  - DoesLockfileExist, [60](#)
  - FileExists, [61](#)
  - FillStudentMapFromFile, [62](#)
  - LOCKFILE, [56](#)
  - STATIC\_USER\_CUMULATIVE\_FILE, [56](#)
  - STATIC\_USER\_DATA\_FILE, [57](#)
  - WriteStudentArrayToFile, [63](#)
- FillStudentMapFromFile
  - Files.c, [51](#)
  - Files.h, [62](#)
- found
  - map\_result, [11](#)
- fullName
  - Student, [12](#)
- GetMemoryPointer
  - memShare.c, [87](#)
  - memShare.h, [92](#)
- gpa
  - Student, [12](#)
- hash\_log2
  - map.c, [69](#)
  - map.h, [79](#)
- hash\_string
  - map.c, [69](#)
  - map.h, [79](#)
- hash\_upperLimit
  - map.c, [70](#)
  - map.h, [80](#)
- HelpCommand
  - Process.c, [95](#)
  - Process.h, [110](#)
- IndicateRereadDone
  - Process.c, [95](#)
  - Process.h, [110](#)
- IndicateRereadNeeded
  - Process.c, [96](#)
- Process.h, [111](#)
- initial\_cumulative\_times
  - Process.c, [104](#)
- Initialize
  - Process.c, [96](#)
  - Process.h, [111](#)
- is\_stopping
  - Process.c, [104](#)
  - Process.h, [119](#)
- IsRereadNeeded
  - Process.c, [97](#)
  - Process.h, [112](#)
- lastLogin
  - Student, [12](#)
- LOCKFILE
  - Files.h, [56](#)
- loginDuration
  - Student, [12](#)
- main
  - main.c, [65](#)
- main.c
  - main, [65](#)
- map, [9](#)
  - buckets, [10](#)
  - size, [10](#)
- map.c
  - \_bucket\_delete, [68](#)
  - \_bucket\_get, [68](#)
  - \_bucket\_insert, [68](#)
  - char\_ratio, [75](#)
  - hash\_log2, [69](#)
  - hash\_string, [69](#)
  - hash\_upperLimit, [70](#)
  - Map\_Delete, [71](#)
  - Map\_Get, [72](#)
  - Map\_Set, [73](#)
  - NewMap, [74](#)
- map.h
  - hash\_log2, [79](#)
  - hash\_string, [79](#)
  - hash\_upperLimit, [80](#)
  - Map\_Delete, [81](#)
  - Map\_Get, [81](#)
  - Map\_Set, [82](#)
  - NewMap, [84](#)
- Map\_Delete
  - map.c, [71](#)
  - map.h, [81](#)
- Map\_Get
  - map.c, [72](#)
  - map.h, [81](#)
- map\_result, [10](#)
  - data, [10](#)
  - found, [11](#)
- Map\_Set
  - map.c, [73](#)
  - map.h, [82](#)



- MEM\_KEY
  - memShare.h, [91](#)
- MEM\_PERMISSIONS
  - memShare.h, [91](#)
- MEM\_SIZE
  - memShare.h, [91](#)
- memShare.c
  - CreateSharedMemory, [87](#)
  - DestroySharedMemory, [87](#)
  - GetMemoryPointer, [87](#)
  - ReleaseMemoryPointer, [88](#)
- memShare.h
  - CreateSharedMemory, [91](#)
  - DestroySharedMemory, [92](#)
  - GetMemoryPointer, [92](#)
  - MEM\_KEY, [91](#)
  - MEM\_PERMISSIONS, [91](#)
  - MEM\_SIZE, [91](#)
  - ReleaseMemoryPointer, [93](#)
- NewMap
  - map.c, [74](#)
  - map.h, [84](#)
- PopulateStudents
  - Build.c, [18](#)
  - Build.h, [31](#)
- Process
  - Process.c, [98](#)
  - Process.h, [113](#)
- Process.c
  - HelpCommand, [95](#)
  - IndicateRereadDone, [95](#)
  - IndicateRereadNeeded, [96](#)
  - initial\_cumulative\_times, [104](#)
  - Initialize, [96](#)
  - is\_stopping, [104](#)
  - IsRereadNeeded, [97](#)
  - Process, [98](#)
  - ResetCommand, [99](#)
  - RunCommand, [100](#)
  - RunHeadless, [101](#)
  - SignalHandle, [102](#)
  - StopCommand, [103](#)
  - student\_map, [104](#)
  - TerminateExistingServer, [103](#)
- Process.h
  - HelpCommand, [110](#)
  - IndicateRereadDone, [110](#)
  - IndicateRereadNeeded, [111](#)
  - Initialize, [111](#)
  - is\_stopping, [119](#)
  - IsRereadNeeded, [112](#)
  - Process, [113](#)
  - ResetCommand, [114](#)
  - RunCommand, [115](#)
  - RunHeadless, [116](#)
  - SignalHandle, [117](#)
  - StopCommand, [118](#)
  - student\_map, [119](#)
  - TerminateExistingServer, [118](#)
- ProcessWhoLine
  - Build.c, [18](#)
  - Build.h, [32](#)
- RandomFlag
  - util.c, [121](#)
  - util.h, [124](#)
- RandomFloat
  - util.c, [121](#)
  - util.h, [124](#)
- RandomInteger
  - util.c, [122](#)
  - util.h, [125](#)
- ReadACP
  - Build.c, [19](#)
  - Build.h, [33](#)
- ReadAcpPipeLine
  - Build.c, [20](#)
  - Build.h, [34](#)
- ReadCumulativeFileLine
  - Build.c, [21](#)
  - Build.h, [35](#)
- ReadInitialCumulative
  - Build.c, [22](#)
  - Build.h, [36](#)
- ReleaseMemoryPointer
  - memShare.c, [88](#)
  - memShare.h, [93](#)
- ResetCommand
  - Process.c, [99](#)
  - Process.h, [114](#)
- RunCommand
  - Process.c, [100](#)
  - Process.h, [115](#)
- RunHeadless
  - Process.c, [101](#)
  - Process.h, [116](#)
- SetAllStudentsInactive
  - Build.c, [23](#)
  - Build.h, [37](#)
- SignalHandle
  - Process.c, [102](#)
  - Process.h, [117](#)
- size
  - map, [10](#)
- src/server/Build.c, [15](#), [26](#)
- src/server/Build.h, [28](#), [40](#)
- src/server/Data.c, [40](#), [42](#)
- src/server/Data.h, [42](#), [45](#)
- src/server/Files.c, [45](#), [53](#)
- src/server/Files.h, [54](#), [64](#)
- src/server/main.c, [64](#), [66](#)
- src/server/map.c, [67](#), [75](#)
- src/server/map.h, [77](#), [85](#)
- src/server/memShare.c, [86](#), [89](#)
- src/server/memShare.h, [89](#), [94](#)

- src/server/Process.c, [94](#), [105](#)
- src/server/Process.h, [109](#), [120](#)
- src/server/util.c, [120](#), [123](#)
- src/server/util.h, [123](#), [125](#)
- STATIC\_USER\_CUMULATIVE\_FILE
  - Files.h, [56](#)
- STATIC\_USER\_DATA\_FILE
  - Files.h, [57](#)
- StopCommand
  - Process.c, [103](#)
  - Process.h, [118](#)
- Student, [11](#)
  - active, [12](#)
  - age, [12](#)
  - fullName, [12](#)
  - gpa, [12](#)
  - lastLogin, [12](#)
  - loginDuration, [12](#)
  - userID, [13](#)
- student\_map
  - Process.c, [104](#)
  - Process.h, [119](#)
- students
  - Build.c, [25](#)
  - Build.h, [39](#)
- TerminateExistingServer
  - Process.c, [103](#)
  - Process.h, [118](#)
- UpdateFromWho
  - Build.c, [23](#)
  - Build.h, [37](#)
- userID
  - Student, [13](#)
- util.c
  - RandomFlag, [121](#)
  - RandomFloat, [121](#)
  - RandomInteger, [122](#)
- util.h
  - RandomFlag, [124](#)
  - RandomFloat, [124](#)
  - RandomInteger, [125](#)
- WriteStudentArrayToFile
  - Files.c, [52](#)
  - Files.h, [63](#)
- WriteStudentsToMemory
  - Build.c, [24](#)
  - Build.h, [38](#)