

calory-counter

1

Generated by Doxygen 1.8.5

Thu Oct 9 2014 11:51:26

## Contents

<b>1</b>	<b>Data Structure Index</b>	<b>1</b>
1.1	Data Structures . . . . .	1
<b>2</b>	<b>File Index</b>	<b>2</b>
2.1	File List . . . . .	2
<b>3</b>	<b>Data Structure Documentation</b>	<b>2</b>
3.1	client_config Struct Reference . . . . .	2
3.1.1	Detailed Description . . . . .	2
3.1.2	Field Documentation . . . . .	3
3.2	food Struct Reference . . . . .	3
3.2.1	Detailed Description . . . . .	3
3.2.2	Field Documentation . . . . .	3
3.3	foodlist Struct Reference . . . . .	4
3.3.1	Detailed Description . . . . .	4
3.3.2	Field Documentation . . . . .	4
3.4	foodlistnode Struct Reference . . . . .	4
3.4.1	Detailed Description . . . . .	5
3.4.2	Field Documentation . . . . .	5
3.5	sockethandler Struct Reference . . . . .	5
3.5.1	Detailed Description . . . . .	5
3.5.2	Field Documentation . . . . .	5
<b>4</b>	<b>File Documentation</b>	<b>6</b>
4.1	diet-client.c File Reference . . . . .	6
4.1.1	Detailed Description . . . . .	7
4.1.2	Function Documentation . . . . .	7
4.2	diet-server.c File Reference . . . . .	8
4.2.1	Detailed Description . . . . .	9
4.2.2	Function Documentation . . . . .	9
4.3	food.c File Reference . . . . .	9
4.3.1	Detailed Description . . . . .	10
4.3.2	Function Documentation . . . . .	11
4.4	food.h File Reference . . . . .	15
4.4.1	Detailed Description . . . . .	16
4.4.2	Function Documentation . . . . .	16
4.5	foodlist.c File Reference . . . . .	20
4.5.1	Detailed Description . . . . .	21
4.5.2	Function Documentation . . . . .	21

4.6	<a href="#">foodlist.h File Reference</a>	24
4.6.1	<a href="#">Detailed Description</a>	24
4.6.2	<a href="#">Function Documentation</a>	25
4.7	<a href="#">foodlistnode.c File Reference</a>	26
4.7.1	<a href="#">Detailed Description</a>	27
4.7.2	<a href="#">Function Documentation</a>	27
4.8	<a href="#">foodlistnode.h File Reference</a>	29
4.8.1	<a href="#">Detailed Description</a>	29
4.8.2	<a href="#">Function Documentation</a>	30
4.9	<a href="#">sock.c File Reference</a>	31
4.9.1	<a href="#">Detailed Description</a>	31
4.9.2	<a href="#">Function Documentation</a>	32
4.10	<a href="#">sock.h File Reference</a>	33
4.10.1	<a href="#">Detailed Description</a>	33
4.10.2	<a href="#">Function Documentation</a>	33
4.11	<a href="#">sockethandler.c File Reference</a>	35
4.11.1	<a href="#">Detailed Description</a>	35
4.11.2	<a href="#">Macro Definition Documentation</a>	36
4.11.3	<a href="#">Function Documentation</a>	36
4.12	<a href="#">sockethandler.h File Reference</a>	37
4.12.1	<a href="#">Detailed Description</a>	37
4.12.2	<a href="#">Function Documentation</a>	37

## [Index](#) 39

# 1 Data Structure Index

## 1.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">client_config</a>	Client config structure	2
<a href="#">food</a>	Food structure for representing a food item	3
<a href="#">foodlist</a>	Foodlist structure for representing a food item	4
<a href="#">foodlistnode</a>	Foodlistnode structure for representing a foodlistnode item	4
<a href="#">sockethandler</a>	Sockethandler structure for representing a sockethandler item	5

## 2 File Index

### 2.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">diet-client.c</a>	Main program file with <a href="#">main()</a> entry point	6
<a href="#">diet-server.c</a>	Main program file with <a href="#">main()</a> entry point	8
<a href="#">food.c</a>	File containing the food structure and its member methods	9
<a href="#">food.h</a>	Header containing the public accessible food methods	15
<a href="#">foodlist.c</a>	File containing the foodlist structure and its member methods	20
<a href="#">foodlist.h</a>	Header containing the public accessible foodlist methods	24
<a href="#">foodlistnode.c</a>	File containing the foodlistnode structure and its member methods	26
<a href="#">foodlistnode.h</a>	Header containing the public accessible foodlistnode methods	29
<a href="#">sock.c</a>	File containing read and write functions for calory socket protocol	31
<a href="#">sock.h</a>	Header file containing read and write functions for calory socket protocol	33
<a href="#">sockethandler.c</a>	File containing the sockethandler structure and its member methods	35
<a href="#">sockethandler.h</a>	Header containing the public accessible sockethandler methods	37

## 3 Data Structure Documentation

### 3.1 client\_config Struct Reference

Client config structure.

#### Data Fields

- char \* [host](#)
- unsigned int [port](#)

#### 3.1.1 Detailed Description

Client config structure.

### 3.1.2 Field Documentation

#### 3.1.2.1 char\* client\_config::host

Hostname to connect to

#### 3.1.2.2 unsigned int client\_config::port

Port to connect to

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

- [diet-client.c](#)

## 3.2 food Struct Reference

food structure for representing a food item

### Data Fields

- char [name](#) [MAX\_NAME\_LEN]
- char [measure](#) [MAX\_NAME\_LEN]
- int [weight](#)
- int [kcal](#)
- int [fat](#)
- int [carbo](#)
- int [protein](#)

### 3.2.1 Detailed Description

food structure for representing a food item

### 3.2.2 Field Documentation

#### 3.2.2.1 int food::carbo

Carbo (g) of the food.

#### 3.2.2.2 int food::fat

Fat (g) of the food.

#### 3.2.2.3 int food::kcal

kCal of the food.

#### 3.2.2.4 char food::measure[MAX\_NAME\_LEN]

Name of the food.

#### 3.2.2.5 char food::name[MAX\_NAME\_LEN]

Name of the food.

#### 3.2.2.6 int food::protein

Protein (g) of the food.

### 3.2.2.7 int food::weight

Weight (g) of the food.

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

- [food.c](#)

## 3.3 foodlist Struct Reference

foodlist structure for representing a food item

### Data Fields

- pthread\_mutex\_t [rw\\_mutex](#)
- pthread\_mutex\_t [r\\_mutex](#)
- int [read\\_count](#)
- [foodlistnode](#) \* [data](#)
- char \* [file](#)

### 3.3.1 Detailed Description

foodlist structure for representing a food item

### 3.3.2 Field Documentation

#### 3.3.2.1 foodlistnode\* foodlist::data

First node of this list

#### 3.3.2.2 char\* foodlist::file

Filename for loading/saving data from/to file

#### 3.3.2.3 pthread\_mutex\_t foodlist::r\_mutex

< Mutex for thread safe write access

#### 3.3.2.4 int foodlist::read\_count

Integer for thread safe read access

#### 3.3.2.5 pthread\_mutex\_t foodlist::rw\_mutex

Mutex for thread safe read-write access

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

- [foodlist.c](#)

## 3.4 foodlistnode Struct Reference

foodlistnode structure for representing a foodlistnode item

#### Data Fields

- [food](#) \* [item](#)
- [foodlistnode](#) \* [next](#)

#### 3.4.1 Detailed Description

foodlistnode structure for representing a foodlistnode item

#### 3.4.2 Field Documentation

##### 3.4.2.1 `food`\* `foodlistnode::item`

Item of this foodlistnode

##### 3.4.2.2 `foodlistnode`\* `foodlistnode::next`

Next foodlistnode of this foodlistnode

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

- [foodlistnode.c](#)

### 3.5 sockethandler Struct Reference

sockethandler structure for representing a sockethandler item

#### Data Fields

- unsigned int [listen\\_port](#)
- pthread\_t [thread\\_pool](#) [[MAX\\_THREADS](#)]
- int [client\\_sockets](#) [[MAX\\_SOCKETS](#)]
- bool [shutdown](#)
- [foodlist](#) \* [foodlist](#)
- pthread\_mutex\_t [mutex](#)
- sem\_t [empty](#)
- sem\_t [full](#)
- size\_t [in](#)
- size\_t [out](#)
- size\_t [count](#)

#### 3.5.1 Detailed Description

sockethandler structure for representing a sockethandler item

#### 3.5.2 Field Documentation

##### 3.5.2.1 `int` `sockethandler::client_sockets`[[MAX\\_SOCKETS](#)]

Array of client sockets for consumer/producer principle

##### 3.5.2.2 `size_t` `sockethandler::count`

number of unconsumed items

### 3.5.2.3 `sem_t sockethandler::empty`

Semaphore to block on empty socket list.

### 3.5.2.4 `foodlist* sockethandler::foodlist`

List of foods to work with

### 3.5.2.5 `sem_t sockethandler::full`

Semaphore to block on full socket list.

### 3.5.2.6 `size_t sockethandler::in`

position of producer in buffer

### 3.5.2.7 `unsigned int sockethandler::listen_port`

Listen port for the server socket

### 3.5.2.8 `pthread_mutex_t sockethandler::mutex`

Mutex to mutual exclude the client\_socket array.

### 3.5.2.9 `size_t sockethandler::out`

position of consumer in buffer

### 3.5.2.10 `bool sockethandler::shutdown`

Flag to notifying all threads to shut down

### 3.5.2.11 `pthread_t sockethandler::thread_pool[MAX_THREADS]`

Thread pool for handling client connections

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

- [sockethandler.c](#)

## 4 File Documentation

### 4.1 diet-client.c File Reference

Main program file with [main\(\)](#) entry point.

```
#include <pthread.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <ctype.h>
#include <unistd.h>
#include <errno.h>
#include "../lib/sock.h"
#include "../lib/food.h"
```



## Data Structures

- struct `client_config`  
*Client config structure.*

## Typedefs

- typedef struct `client_config` `client_config`  
*Forward declaration of `client_config`.*

## Functions

- void `usage` (char \*pname)  
*Prints the help for diet-client to the console.*
- bool `ask_user_cancel` ()  
*Method for getting a true/false user input.*
- `food *` `get_food_from_user` ()  
*Method for creating a food object with help of user input.*
- void `client_loop` (`client_config` \*c)  
*Loop function with handles the client connection and user input stuff.*
- int `main` (int argc, char \*\*argv)  
*Main entry point of diet-client.*

## Variables

- bool `client_exit` = false  
*This is set to exit when the application should exit gracefully.*

### 4.1.1 Detailed Description

Main program file with `main()` entry point.

#### Author

Lukas Elsner

#### Date

01-09-2014 The calory-counter client application.

### 4.1.2 Function Documentation

#### 4.1.2.1 bool ask\_user\_cancel ( )

Method for getting a true/false user input.

#### Returns

True, if the user decided to cancel, false if the user wants to continue

#### 4.1.2.2 void client\_loop ( client\_config \* c )

Loop function with handles the client connection and user input stuff.

**Parameters**

<i>client_config*</i>	A pointer to the client configuration
-----------------------	---------------------------------------

**4.1.2.3 food\* get\_food\_from\_user ( )**

Method for creating a food object with help of user input.

**Returns**

A pointer to a food structure if the user entered all data correctly, NULL if the user decided to cancel the operation.

**4.1.2.4 int main ( int argc, char \*\* argv )**

Main entry point of diet-client.

**Parameters**

<i>int</i>	Number of arguments
<i>char**</i>	Pointer to array of arguments

**Returns**

Exit code of diet-client

**4.1.2.5 void usage ( char \* pname )**

Prints the help for diet-client to the console.

**Parameters**

<i>char*</i>	Program name
--------------	--------------

**4.2 diet-server.c File Reference**

Main program file with [main\(\)](#) entry point.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include "sockethandler.h"
```

**Functions**

- void [usage](#) (char \*pname)  
*Prints the help for diet-server to the console.*
- void [signal\\_callback\\_handler](#) (int signum)  
*Define the function to be called when ctrl-c (SIGINT) signal is sent to process.*
- int [main](#) (int argc, char \*\*argv)  
*Main entry point of airport-sim.*

## Variables

- `foodlist * fl`  
*Representation of the food list.*
- `sockethandler * s`  
*Representation of the socket handler.*

### 4.2.1 Detailed Description

Main program file with `main()` entry point.

#### Author

Lukas Elsner

#### Date

01-09-2014 The calory-counter server application.

### 4.2.2 Function Documentation

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

Main entry point of airport-sim.

##### Parameters

<code>int</code>	Number of arguments
<code>char**</code>	Pointer to array of arguments

##### Returns

Exit code of airport-sim

#### 4.2.2.2 `void usage ( char * pname )`

Prints the help for diet-server to the console.

##### Parameters

<code>char*</code>	Program name
--------------------	--------------

## 4.3 food.c File Reference

File containing the food structure and its member methods.

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

## Data Structures

- struct `food`  
*food structure for representing a food item*

## Functions

- `food * food_init ()`  
*Constructor for food.*
- `void food_set_name (food *f, const char *name)`  
*Method for setting the name of a food structure.*
- `char * food_get_name (food *f)`  
*Method for getting the name of a food structure.*
- `void food_set_measure (food *f, const char *measure)`  
*Method for setting the measure of a food structure.*
- `char * food_get_measure (food *f)`  
*Method for getting the measure of a food structure.*
- `char * food_to_string (food *f)`  
*Method for getting a string representation of a food structure.*
- `void food_set_weight (food *f, const int weight)`  
*Method for setting the weight of a food structure.*
- `int food_get_weight (food *f)`  
*Method for getting the weight of a food structure.*
- `void food_set_kcal (food *f, const int kcal)`  
*Method for setting the kCal of a food structure.*
- `int food_get_kcal (food *f)`  
*Method for getting the kCal of a food structure.*
- `void food_set_fat (food *f, const int fat)`  
*Method for setting the fat of a food structure.*
- `int food_get_fat (food *f)`  
*Method for getting the fat of a food structure.*
- `void food_set_carbo (food *f, const int carbo)`  
*Method for setting the carbo of a food structure.*
- `int food_get_carbo (food *f)`  
*Method for getting the carbo of a food structure.*
- `void food_set_protein (food *f, const int protein)`  
*Method for setting the protein of a food structure.*
- `int food_get_protein (food *f)`  
*Method for getting the protein of a food structure.*
- `char * food_serialize (food *f)`  
*Method for serializing a food structure into a character array.*
- `food * food_deserialize (char *c)`  
*Method for deserializing a char array into a food structure, this method is being used for loading foods from the csv file, as well as for the network communication.*
- `size_t food_get_size ()`  
*Method for getting the size of a food structure.*
- `void food_destroy (food *f)`  
*Destructor for food.*

### 4.3.1 Detailed Description

File containing the food structure and its member methods.

#### Author

Lukas Elsner

#### Date

01-09-2014

## 4.3.2 Function Documentation

## 4.3.2.1 food\* food\_deserialize ( char \* )

Method for deserializing a char array into a food structure, this method is being used for loading foods from the csv file, as well as for the network communication.

## Parameters

<i>char*</i>	The serialized structure
--------------	--------------------------

## Returns

A pointer to the deserialized structure. Must be freed with [food\\_destroy\(food \\*\)](#)

## 4.3.2.2 void food\_destroy ( food \* )

Destructor for food.

## Parameters

<i>food*</i>	Pointer to structure to be freed
--------------	----------------------------------

## 4.3.2.3 int food\_get\_carbo ( food \* )

Method for getting the carbo of a food structure.

## Parameters

<i>food*</i>	Pointer to structure to work on
--------------	---------------------------------

## Returns

Carbo value in g of the food

## 4.3.2.4 int food\_get\_fat ( food \* )

Method for getting the fat of a food structure.

## Parameters

<i>food*</i>	Pointer to structure to work on
--------------	---------------------------------

## Returns

Fat value in g of the food

## 4.3.2.5 int food\_get\_kcal ( food \* )

Method for getting the kCal of a food structure.

## Parameters

<i>food*</i>	Pointer to structure to work on
--------------	---------------------------------

## Returns

kCal value of the food

## 4.3.2.6 char\* food\_get\_measure ( food \* )

Method for getting the measure of a food structure.

**Parameters**

<i>food*</i>	Pointer to structure to work on
--------------	---------------------------------

**Returns**

A pointer to a string containing the measure of the food. Must be freed by caller.

**4.3.2.7 char\* food\_get\_name ( food \* )**

Method for getting the name of a food structure.

**Parameters**

<i>food*</i>	Pointer to structure to work on
--------------	---------------------------------

**Returns**

A pointer to a string containing the name of the food. Must be freed by caller.

**4.3.2.8 int food\_get\_protein ( food \* )**

Method for getting the protein of a food structure.

**Parameters**

<i>food*</i>	Pointer to structure to work on
--------------	---------------------------------

**Returns**

Protein value in g of the food

**4.3.2.9 size\_t food\_get\_size ( )**

Method for getting the size of a food structure.

**Returns**

Size of a food structure

**4.3.2.10 int food\_get\_weight ( food \* )**

Method for getting the weight of a food structure.

**Parameters**

<i>food*</i>	Pointer to structure to work on
--------------	---------------------------------

**Returns**

Weight value in g of the food

**4.3.2.11 food\* food\_init ( )**

Constructor for food.

**Returns**

A pointer to the food structure, representing the created object

After using this structure, it must be freed with [food\\_destroy\(food \\*\)](#)

#### 4.3.2.12 char\* food\_serialize ( food \* )

Method for serializing a food structure into a character array.

**Parameters**

<i>food*</i>	Pointer to structure to work on
--------------	---------------------------------

**Returns**

A pointer to the serialized structure. Must be freed by caller

**4.3.2.13 void food\_set\_carbo ( food \*, const int )**

Method for setting the carbo of a food structure.

**Parameters**

<i>food*</i>	Pointer to structure to work on
<i>int</i>	Carbo value in g to be set

**4.3.2.14 void food\_set\_fat ( food \*, const int )**

Method for setting the fat of a food structure.

**Parameters**

<i>food*</i>	Pointer to structure to work on
<i>int</i>	Fat value in g to be set

**4.3.2.15 void food\_set\_kcal ( food \*, const int )**

Method for setting the kCal of a food structure.

**Parameters**

<i>food*</i>	Pointer to structure to work on
<i>int</i>	kCal value to be set

**4.3.2.16 void food\_set\_measure ( food \*, const char \* )**

Method for setting the measure of a food structure.

**Parameters**

<i>food*</i>	Pointer to structure to work on
<i>char*</i>	Pointer to character array containing the measure to be set

**4.3.2.17 void food\_set\_name ( food \*, const char \* )**

Method for setting the name of a food structure.

**Parameters**

<i>food*</i>	Pointer to structure to work on
<i>char*</i>	Pointer to character array containing the name to be set

**4.3.2.18 void food\_set\_protein ( food \*, const int )**

Method for setting the protein of a food structure.



## Parameters

<i>food*</i>	Pointer to structure to work on
<i>int</i>	Protein value in g to be set

## 4.3.2.19 void food\_set\_weight ( food \*, const int )

Method for setting the weight of a food structure.

## Parameters

<i>food*</i>	Pointer to structure to work on
<i>int</i>	Weight value in g to be set

## 4.3.2.20 char\* food\_to\_string ( food \* )

Method for getting a string representation of a food structure.

## Parameters

<i>food*</i>	Pointer to structure to work on
--------------	---------------------------------

## Returns

String representation of food structure. Must be freed by caller.

## 4.4 food.h File Reference

Header containing the public accessible food methods.

## Macros

- #define **MAX\_NAME\_LEN** 1024
- #define **MAX\_MEASURE\_LEN** 256

## Typedefs

- typedef struct **food** **food**  
*Forward declaration for food.*

## Functions

- **food \*** **food\_init** ()  
*Constructor for food.*
- char \* **food\_serialize** (**food \***)  
*Method for serializing a food structure into a character array.*
- **food \*** **food\_deserialize** (char \*)  
*Method for deserializing a char array into a food structure, this method is being used for loading foods from the csv file, as well as for the network communication.*
- void **food\_set\_name** (**food \***, const char \*)  
*Method for setting the name of a food structure.*
- char \* **food\_get\_name** (**food \***)  
*Method for getting the name of a food structure.*
- void **food\_set\_measure** (**food \***, const char \*)  
*Method for setting the measure of a food structure.*

- char \* `food_get_measure` (food \*)  
*Method for getting the measure of a food structure.*
- void `food_set_weight` (food \*, const int)  
*Method for setting the weight of a food structure.*
- int `food_get_weight` (food \*)  
*Method for getting the weight of a food structure.*
- void `food_set_kcal` (food \*, const int)  
*Method for setting the kCal of a food structure.*
- int `food_get_kcal` (food \*)  
*Method for getting the kCal of a food structure.*
- void `food_set_fat` (food \*, const int)  
*Method for setting the fat of a food structure.*
- int `food_get_fat` (food \*)  
*Method for getting the fat of a food structure.*
- void `food_set_carbo` (food \*, const int)  
*Method for setting the carbo of a food structure.*
- int `food_get_carbo` (food \*)  
*Method for getting the carbo of a food structure.*
- void `food_set_protein` (food \*, const int)  
*Method for setting the protein of a food structure.*
- int `food_get_protein` (food \*)  
*Method for getting the protein of a food structure.*
- char \* `food_to_string` (food \*)  
*Method for getting a string representation of a food structure.*
- size\_t `food_get_size` ()  
*Method for getting the size of a food structure.*
- void `food_destroy` (food \*)  
*Destructor for food.*

#### 4.4.1 Detailed Description

Header containing the public accessible food methods.

##### Author

Lukas Elsner

##### Date

25-09-2014

#### 4.4.2 Function Documentation

##### 4.4.2.1 food\* food\_deserialize ( char \* )

Method for deserializing a char array into a food structure, this method is being used for loading foods from the csv file, as well as for the network communication.

##### Parameters

---

<i>char*</i>	The serialized structure
--------------	--------------------------

**Returns**

A pointer to the deserialized structure. Must be freed with [food\\_destroy\(food \\*\)](#)

**4.4.2.2 void food\_destroy ( food \* )**

Destructor for food.

**Parameters**

<i>food*</i>	Pointer to structure to be freed
--------------	----------------------------------

**4.4.2.3 int food\_get\_carbo ( food \* )**

Method for getting the carbo of a food structure.

**Parameters**

<i>food*</i>	Pointer to structure to work on
--------------	---------------------------------

**Returns**

Carbo value in g of the food

**4.4.2.4 int food\_get\_fat ( food \* )**

Method for getting the fat of a food structure.

**Parameters**

<i>food*</i>	Pointer to structure to work on
--------------	---------------------------------

**Returns**

Fat value in g of the food

**4.4.2.5 int food\_get\_kcal ( food \* )**

Method for getting the kCal of a food structure.

**Parameters**

<i>food*</i>	Pointer to structure to work on
--------------	---------------------------------

**Returns**

kCal value of the food

**4.4.2.6 char\* food\_get\_measure ( food \* )**

Method for getting the measure of a food structure.

**Parameters**

<i>food*</i>	Pointer to structure to work on
--------------	---------------------------------

**Returns**

A pointer to a string containing the measure of the food. Must be freed by caller.

**4.4.2.7 char\* food\_get\_name ( food \* )**

Method for getting the name of a food structure.

**Parameters**

<i>food*</i>	Pointer to structure to work on
--------------	---------------------------------

**Returns**

A pointer to a string containing the name of the food. Must be freed by caller.

**4.4.2.8 int food\_get\_protein ( food \* )**

Method for getting the protein of a food structure.

**Parameters**

<i>food*</i>	Pointer to structure to work on
--------------	---------------------------------

**Returns**

Protein value in g of the food

**4.4.2.9 size\_t food\_get\_size ( )**

Method for getting the size of a food structure.

**Returns**

Size of a food structure

**4.4.2.10 int food\_get\_weight ( food \* )**

Method for getting the weight of a food structure.

**Parameters**

<i>food*</i>	Pointer to structure to work on
--------------	---------------------------------

**Returns**

Weight value in g of the food

**4.4.2.11 food\* food\_init ( )**

Constructor for food.

**Returns**

A pointer to the food structure, representing the created object

After using this structure, it must be freed with [food\\_destroy\(food \\*\)](#)

**4.4.2.12 char\* food\_serialize ( food \* )**

Method for serializing a food structure into a character array.

## Parameters

<i>food*</i>	Pointer to structure to work on
--------------	---------------------------------

## Returns

A pointer to the serialized structure. Must be freed by caller

## 4.4.2.13 void food\_set\_carbo ( food \*, const int )

Method for setting the carbo of a food structure.

## Parameters

<i>food*</i>	Pointer to structure to work on
<i>int</i>	Carbo value in g to be set

## 4.4.2.14 void food\_set\_fat ( food \*, const int )

Method for setting the fat of a food structure.

## Parameters

<i>food*</i>	Pointer to structure to work on
<i>int</i>	Fat value in g to be set

## 4.4.2.15 void food\_set\_kcal ( food \*, const int )

Method for setting the kCal of a food structure.

## Parameters

<i>food*</i>	Pointer to structure to work on
<i>int</i>	kCal value to be set

## 4.4.2.16 void food\_set\_measure ( food \*, const char \* )

Method for setting the measure of a food structure.

## Parameters

<i>food*</i>	Pointer to structure to work on
<i>char*</i>	Pointer to character array containing the measure to be set

## 4.4.2.17 void food\_set\_name ( food \*, const char \* )

Method for setting the name of a food structure.

## Parameters

<i>food*</i>	Pointer to structure to work on
<i>char*</i>	Pointer to character array containing the name to be set

## 4.4.2.18 void food\_set\_protein ( food \*, const int )

Method for setting the protein of a food structure.

**Parameters**

<i>food*</i>	Pointer to structure to work on
<i>int</i>	Protein value in g to be set

**4.4.2.19 void food\_set\_weight ( food \*, const int )**

Method for setting the weight of a food structure.

**Parameters**

<i>food*</i>	Pointer to structure to work on
<i>int</i>	Weight value in g to be set

**4.4.2.20 char\* food\_to\_string ( food \* )**

Method for getting a string representation of a food structure.

**Parameters**

<i>food*</i>	Pointer to structure to work on
--------------	---------------------------------

**Returns**

String representation of food structure. Must be freed by caller.

**4.5 foodlist.c File Reference**

File containing the foodlist structure and its member methods.

```
#include <stdlib.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <strings.h>
#include <pthread.h>
#include "food.h"
#include "foodlistnode.h"
#include "foodlist.h"
```

**Data Structures**

- struct [foodlist](#)  
*foodlist structure for representing a food item*

**Functions**

- void [start\\_read](#) (foodlist \*fl)  
*Helper function to enter a critical section for reading.*
- void [end\\_read](#) (foodlist \*fl)  
*Helper function to exit a critical section for reading.*
- void [start\\_write](#) (foodlist \*fl)  
*Helper function to enter a critical section for writing.*
- void [end\\_write](#) (foodlist \*fl)  
*Helper function to exit a critical section for writing.*

- int `cmpfunc` (const void \*a, const void \*b)  
*Compare function for using qsort() with food objects.*
- foodlist \* `foodlist_init` ()  
*Constructor for foodlist.*
- foodlist \* `foodlist_init_csv` (char \*file)  
*Constructor for foodlist, loads a csv file which is passed as argument, throws a warning if file cannot be read and starts with empty dataset then.*
- int `foodlist_count` (foodlist \*fl)  
*Method for getting the length of the list.*
- bool `foodlist_is_empty` (foodlist \*fl)  
*Method for checking if the foodlist is empty.*
- void `foodlist_append` (foodlist \*fl, food \*\*f)  
*Method for appending a food structure to the list.*
- foodlistnode \* `foodlist_get_data` (foodlist \*fl)  
*Method for getting the data of the list.*
- food \*\* `foodlist_find` (foodlist \*fl, char \*str, size\_t \*num)  
*Method for finding food within the food list.*
- void `foodlist_save` (foodlist \*fl)  
*Method for saving the food structure to a file.*
- void `foodlist_destroy` (foodlist \*fl)  
*Destructor for foodlist.*

#### 4.5.1 Detailed Description

File containing the foodlist structure and its member methods.

##### Author

Lukas Elsner

##### Date

25-09-2014

#### 4.5.2 Function Documentation

##### 4.5.2.1 int cmpfunc ( const void \* a, const void \* b )

Compare function for using qsort() with food objects.

##### Parameters

<i>void*</i>	Pointer to first food object
<i>void*</i>	Pointer to second food object

##### Returns

An integer less than, equal to, or greater than zero if a (or the first n bytes thereof) is found, respectively, to be less than, to match, or be greater than b

##### 4.5.2.2 void end\_read ( foodlist \* fl )

Helper function to exit a critical section for reading.

**Parameters**

<i>foodlist*</i>	The foodlist structure to unlock
------------------	----------------------------------

**4.5.2.3 void end\_write ( foodlist \* fl )**

Helper function to exit a critical section for writing.

**Parameters**

<i>foodlist*</i>	The foodlist structure to unlock
------------------	----------------------------------

**4.5.2.4 void foodlist\_append ( foodlist \* , food \*\* )**

Method for appending a food structure to the list.

**Parameters**

<i>foodlist*</i>	Pointer to structure to work on
<i>food**</i>	pointer to pointer to food structure to add

**4.5.2.5 int foodlist\_count ( foodlist \* )**

Method for getting the length of the list.

**Parameters**

<i>foodlist*</i>	Pointer to structure to work on
------------------	---------------------------------

**Returns**

Length of the list

**4.5.2.6 void foodlist\_destroy ( foodlist \* )**

Destructor for foodlist.

**Parameters**

<i>foodlist*</i>	Pointer to structure to be freed
------------------	----------------------------------

**4.5.2.7 food\*\* foodlist\_find ( foodlist \* , char \* , size\_t \* )**

Method for finding food within the food list.

**Parameters**

<i>foodlist*</i>	Pointer to structure to work on
<i>char*</i>	A pointer to the string which should be found
<i>size_t*</i>	Pointer to a size_t instance. The method updates its value to the length of the returned list.

**Returns**

food\*\* A pointer to an array of food pointers, which are satisfying the search criteria.

**4.5.2.8 foodlistnode\* foodlist\_get\_data ( foodlist \* )**

Method for getting the data of the list.



**Parameters**

<i>foodlist*</i>	Pointer to structure to work on
------------------	---------------------------------

**Returns**

First node of the list.

**4.5.2.9 foodlist\* foodlist\_init ( )**

Constructor for foodlist.

**Returns**

A pointer to the foodlist structure, representing the created object

After using this structure, it must be freed with [foodlist\\_destroy\(foodlist \\*\)](#)

**4.5.2.10 foodlist\* foodlist\_init\_csv ( char \* )**

Constructor for foodlist, loads a csv file which is passed as argument, throws a warning if file cannot be read and starts with empty dataset then.

**Parameters**

<i>char*</i>	Filename to the csv-file to be loaded.
--------------	--

**Returns**

A pointer to the foodlist structure, representing the created object

After using this structure, it must be freed with [foodlist\\_destroy\(foodlist \\*\)](#)

**4.5.2.11 bool foodlist\_is\_empty ( foodlist \* )**

Method for checking if the foodlist is empty.

**Parameters**

<i>foodlist*</i>	Pointer to structure to work on
------------------	---------------------------------

**Returns**

True, if the foodlist is empty, false otherwise

**4.5.2.12 void foodlist\_save ( foodlist \* )**

Method for saving the food structure to a file.

**Parameters**

<i>foodlist*</i>	Pointer to structure to work on
------------------	---------------------------------

Before saving, the foodlist is being sorted by the name of the foods. If the file does not exist, it will be created.

**4.5.2.13 void start\_read ( foodlist \* fl )**

Helper function to enter a critical section for reading.

## Parameters

<i>foodlist*</i>	The foodlist structure to lock
------------------	--------------------------------

## 4.5.2.14 void start\_write ( foodlist \* fl )

Helper function to enter a critical section for writing.

## Parameters

<i>foodlist*</i>	The foodlist structure to lock
------------------	--------------------------------

## 4.6 foodlist.h File Reference

Header containing the public accessible foodlist methods.

```
#include "food.h"
#include "foodlistnode.h"
```

## Typedefs

- typedef struct [foodlist](#) [foodlist](#)  
*Forward declaration for foodlist.*

## Functions

- [foodlist \\*](#) [foodlist\\_init](#) ()  
*Constructor for foodlist.*
- [foodlist \\*](#) [foodlist\\_init\\_csv](#) (char \*)  
*Constructor for foodlist, loads a csv file which is passed as argument, throws a warning if file cannot be read and starts with empty dataset then.*
- void [foodlist\\_append](#) ([foodlist \\*](#), [food \\*\\*](#))  
*Method for appending a food structure to the list.*
- [food \\*\\*](#) [foodlist\\_find](#) ([foodlist \\*](#), char \*, [size\\_t \\*](#))  
*Method for finding food within the food list.*
- void [foodlist\\_save](#) ([foodlist \\*](#))  
*Method for saving the food structure to a file.*
- int [foodlist\\_count](#) ([foodlist \\*](#))  
*Method for getting the length of the list.*
- [foodlistnode \\*](#) [foodlist\\_get\\_data](#) ([foodlist \\*](#))  
*Method for getting the data of the list.*
- bool [foodlist\\_is\\_empty](#) ([foodlist \\*](#))  
*Method for checking if the foodlist is empty.*
- void [foodlist\\_destroy](#) ([foodlist \\*](#))  
*Destructor for foodlist.*

## 4.6.1 Detailed Description

Header containing the public accessible foodlist methods.

## Author

Lukas Elsner

## Date

25-09-2014

## 4.6.2 Function Documentation

## 4.6.2.1 void foodlist\_append ( foodlist \* , food \*\* )

Method for appending a food structure to the list.

## Parameters

<i>foodlist*</i>	Pointer to structure to work on
<i>food**</i>	pointer to pointer to food structure to add

## 4.6.2.2 int foodlist\_count ( foodlist \* )

Method for getting the length of the list.

## Parameters

<i>foodlist*</i>	Pointer to structure to work on
------------------	---------------------------------

## Returns

Length of the list

## 4.6.2.3 void foodlist\_destroy ( foodlist \* )

Destructor for foodlist.

## Parameters

<i>foodlist*</i>	Pointer to structure to be freed
------------------	----------------------------------

## 4.6.2.4 food\*\* foodlist\_find ( foodlist \* , char \* , size\_t \* )

Method for finding food within the food list.

## Parameters

<i>foodlist*</i>	Pointer to structure to work on
<i>char*</i>	A pointer to the string which should be found
<i>size_t*</i>	Pointer to a size_t instance. The method updates its value to the length of the returned list.

## Returns

food\*\* A pointer to an array of food pointers, which are satisfying the search criteria.

## 4.6.2.5 foodlistnode\* foodlist\_get\_data ( foodlist \* )

Method for getting the data of the list.

**Parameters**

<i>foodlist*</i>	Pointer to structure to work on
------------------	---------------------------------

**Returns**

First node of the list.

**4.6.2.6 foodlist\* foodlist\_init ( )**

Constructor for foodlist.

**Returns**

A pointer to the foodlist structure, representing the created object

After using this structure, it must be freed with [foodlist\\_destroy\(foodlist \\*\)](#)

**4.6.2.7 foodlist\* foodlist\_init\_csv ( char \* )**

Constructor for foodlist, loads a csv file which is passed as argument, throws a warning if file cannot be read and starts with empty dataset then.

**Parameters**

<i>char*</i>	Filename to the csv-file to be loaded.
--------------	--

**Returns**

A pointer to the foodlist structure, representing the created object

After using this structure, it must be freed with [foodlist\\_destroy\(foodlist \\*\)](#)

**4.6.2.8 bool foodlist\_is\_empty ( foodlist \* )**

Method for checking if the foodlist is empty.

**Parameters**

<i>foodlist*</i>	Pointer to structure to work on
------------------	---------------------------------

**Returns**

True, if the foodlist is empty, false otherwise

**4.6.2.9 void foodlist\_save ( foodlist \* )**

Method for saving the food structure to a file.

**Parameters**

<i>foodlist*</i>	Pointer to structure to work on
------------------	---------------------------------

Before saving, the foodlist is being sorted by the name of the foods. If the file does not exist, it will be created.

**4.7 foodlistnode.c File Reference**

File containing the foodlistnode structure and its member methods.

```
#include <time.h>
#include <stdlib.h>
#include <assert.h>
#include "foodlistnode.h"
```

## Data Structures

- struct `foodlistnode`  
*foodlistnode structure for representing a foodlistnode item*

## Functions

- `foodlistnode * foodlistnode_init ()`  
*constructor for foodlistnode*
- `foodlistnode * foodlistnode_get_next (foodlistnode *fln)`  
*Method for getting the next foodlistnode of a foodlistnode structure.*
- `food * foodlistnode_get_item (foodlistnode *fln)`  
*Method for getting the item of a foodlistnode structure.*
- `void foodlistnode_set_next (foodlistnode *fln, foodlistnode **f)`  
*Method for setting the next node of a node.*
- `void foodlistnode_set_item (foodlistnode *fln, food **f)`  
*Method for setting the item of a node.*
- `bool foodlistnode_has_next (foodlistnode *fln)`  
*Method for checking if a foodlistnode has a next element.*
- `int foodlistnode_count (foodlistnode *fln)`  
*Recursive method for getting the number of items in this foodlistnode structure.*
- `void foodlistnode_destroy (foodlistnode *fln)`  
*Destructor for foodlistnode.*

### 4.7.1 Detailed Description

File containing the foodlistnode structure and its member methods.

#### Author

Lukas Elsner

#### Date

01-09-2014

### 4.7.2 Function Documentation

#### 4.7.2.1 `int foodlistnode_count ( foodlistnode * )`

Recursive method for getting the number of items in this foodlistnode structure.

##### Parameters

<i>foodlistnode*</i>	Pointer to structure to work on
----------------------	---------------------------------

##### Returns

The number of items in this foodlistnode structure

#### 4.7.2.2 `void foodlistnode_destroy ( foodlistnode * )`

Destructor for foodlistnode.

**Parameters**

<i>foodlistnode*</i>	Pointer to structure to be freed
----------------------	----------------------------------

**4.7.2.3 food\* foodlistnode\_get\_item ( foodlistnode \* fln )**

Method for getting the item of a foodlistnode structure.

**Parameters**

<i>foodlistnode*</i>	Pointer to structure to work on
----------------------	---------------------------------

**Returns**

Item of the passed foodlistnode

**4.7.2.4 foodlistnode\* foodlistnode\_get\_next ( foodlistnode \* fln )**

Method for getting the next foodlistnode of a foodlistnode structure.

**Parameters**

<i>foodlistnode*</i>	Pointer to structure to work on
----------------------	---------------------------------

**Returns**

Next foodlistnode of the passed foodlistnode

**4.7.2.5 bool foodlistnode\_has\_next ( foodlistnode \* )**

Method for checking if a foodlistnode has a next element.

**Parameters**

<i>foodlistnode*</i>	Pointer to structure to work on
----------------------	---------------------------------

**Returns**

True, if passed foodlistnode has a next element, false otherwise

**4.7.2.6 foodlistnode\* foodlistnode\_init ( )**

constructor for foodlistnode

**Returns**

A pointer to the foodlistnode structure, representing the created object

After using this structure, it must be freed with [foodlistnode\\_destroy\(foodlistnode \\*\)](#)

**4.7.2.7 void foodlistnode\_set\_item ( foodlistnode \* fln, food \*\* f )**

Method for setting the item of a node.

**Parameters**

<i>foodlistnode*</i>	Pointer to structure to work on
----------------------	---------------------------------

<code>foodlistnode**</code>	Pointer to pointer to food to set
-----------------------------	-----------------------------------

#### 4.7.2.8 void foodlistnode\_set\_next ( foodlistnode \* fln, foodlistnode \*\* f )

Method for setting the next node of a node.

##### Parameters

<code>foodlistnode*</code>	Pointer to structure to work on
<code>foodlistnode**</code>	Pointer to pointer to foodlistnode to add

## 4.8 foodlistnode.h File Reference

Header containing the public accessible foodlistnode methods.

```
#include <stdbool.h>
#include "food.h"
```

### Typedefs

- typedef struct [foodlistnode](#) [foodlistnode](#)  
*Forward declaration for foodlistnode.*

### Functions

- [foodlistnode \\*](#) [foodlistnode\\_init](#) ()  
*constructor for foodlistnode*
- void [foodlistnode\\_set\\_next](#) ([foodlistnode \\*](#)fln, [foodlistnode \\*\\*](#)f)  
*Method for setting the next node of a node.*
- [foodlistnode \\*](#) [foodlistnode\\_get\\_next](#) ([foodlistnode \\*](#)fln)  
*Method for getting the next foodlistnode of a foodlistnode structure.*
- void [foodlistnode\\_set\\_item](#) ([foodlistnode \\*](#)fln, [food \\*\\*](#)f)  
*Method for setting the item of a node.*
- [food \\*](#) [foodlistnode\\_get\\_item](#) ([foodlistnode \\*](#)fln)  
*Method for getting the item of a foodlistnode structure.*
- int [foodlistnode\\_count](#) ([foodlistnode \\*](#))  
*Recursive method for getting the number of items in this foodlistnode structure.*
- bool [foodlistnode\\_has\\_next](#) ([foodlistnode \\*](#))  
*Method for checking if a foodlistnode has a next element.*
- void [foodlistnode\\_destroy](#) ([foodlistnode \\*](#))  
*Destructor for foodlistnode.*

#### 4.8.1 Detailed Description

Header containing the public accessible foodlistnode methods.

##### Author

Lukas Elsner

##### Date

25-09-2014

## 4.8.2 Function Documentation

### 4.8.2.1 `int foodlistnode_count ( foodlistnode * )`

Recursive method for getting the number of items in this foodlistnode structure.

#### Parameters

<i>foodlistnode*</i>	Pointer to structure to work on
----------------------	---------------------------------

#### Returns

The number of items in this foodlistnode structure

### 4.8.2.2 `void foodlistnode_destroy ( foodlistnode * )`

Destructor for foodlistnode.

#### Parameters

<i>foodlistnode*</i>	Pointer to structure to be freed
----------------------	----------------------------------

### 4.8.2.3 `food* foodlistnode_get_item ( foodlistnode * fln )`

Method for getting the item of a foodlistnode structure.

#### Parameters

<i>foodlistnode*</i>	Pointer to structure to work on
----------------------	---------------------------------

#### Returns

Item of the passed foodlistnode

### 4.8.2.4 `foodlistnode* foodlistnode_get_next ( foodlistnode * fln )`

Method for getting the next foodlistnode of a foodlistnode structure.

#### Parameters

<i>foodlistnode*</i>	Pointer to structure to work on
----------------------	---------------------------------

#### Returns

Next foodlistnode of the passed foodlistnode

### 4.8.2.5 `bool foodlistnode_has_next ( foodlistnode * )`

Method for checking if a foodlistnode has a next element.

#### Parameters

<i>foodlistnode*</i>	Pointer to structure to work on
----------------------	---------------------------------

#### Returns

True, if passed foodlistnode has a next element, false otherwise

### 4.8.2.6 `foodlistnode* foodlistnode_init ( )`

constructor for foodlistnode



## Returns

A pointer to the foodlistnode structure, representing the created object

After using this structure, it must be freed with `foodlistnode_destroy(foodlistnode *)`

## 4.8.2.7 void foodlistnode\_set\_item ( foodlistnode \* fln, food \*\* f )

Method for setting the item of a node.

## Parameters

<i>foodlistnode*</i>	Pointer to structure to work on
<i>foodlistnode**</i>	Pointer to pointer to food to set

## 4.8.2.8 void foodlistnode\_set\_next ( foodlistnode \* fln, foodlistnode \*\* f )

Method for setting the next node of a node.

## Parameters

<i>foodlistnode*</i>	Pointer to structure to work on
<i>foodlistnode**</i>	Pointer to pointer to foodlistnode to add

## 4.9 sock.c File Reference

File containing read and write functions for calory socket protocol.

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include "sock.h"
```

## Functions

- bool `sock_write` (int socket, char \*data)  
*Lower level function to send data to the other endpoint.*
- size\_t `sock_read` (int socket, char \*data)  
*Function to read data from the other endpoint.*
- bool `sock_send_food` (int socket, char \*data)  
*Higher level function to send serialized food to the other endpoint.*
- bool `sock_send_search` (int socket, char \*data)  
*Higher level function to send a search request to the other endpoint.*
- bool `sock_send_count` (int socket, char \*data)  
*Higher level function to send the number of found items to the other endpoint.*

## 4.9.1 Detailed Description

File containing read and write functions for calory socket protocol.

## Author

Lukas Elsner

## Date

02-09-2014

## 4.9.2 Function Documentation

4.9.2.1 `size_t sock_read ( int socket, char * data )`

Function to read data from the other endpoint.

## Parameters

<i>int</i>	The socket to communicate with
<i>char*</i>	A pointer to a buffer which for the read data. Must be at least 4096 bytes long.

## Returns

The size of read data

4.9.2.2 `bool sock_send_count ( int socket, char * data )`

Higher level function to send the number of found items to the other endpoint.

## Parameters

<i>int</i>	The socket to communicate with
<i>char*</i>	The number of found items as string value

## Returns

True, if the communication was successful, false otherwise

4.9.2.3 `bool sock_send_food ( int socket, char * data )`

Higher level function to send serialized food to the other endpoint.

## Parameters

<i>int</i>	The socket to communicate with
<i>char*</i>	The serialized food to send

## Returns

True, if the communication was successful, false otherwise

4.9.2.4 `bool sock_send_search ( int socket, char * data )`

Higher level function to send a search request to the other endpoint.

## Parameters

<i>int</i>	The socket to communicate with
<i>char*</i>	The search term to send

## Returns

True, if the communication was successful, false otherwise

4.9.2.5 `bool sock_write ( int socket, char * data )`

Lower level function to send data to the other endpoint.

## Parameters

<i>int</i>	The socket to communicate with
<i>char*</i>	The data to send

## Returns

True, if the communication was successful, false otherwise

## 4.10 sock.h File Reference

Header file containing read and write functions for calory socket protocol.

```
#include <stdbool.h>
```

## Macros

- #define **BUF\_LEN** 4096
- #define **RE\_LEN** 32

## Functions

- bool [sock\\_write](#) (int socket, char \*data)  
*Lower level function to send data to the other endpoint.*
- size\_t [sock\\_read](#) (int socket, char \*data)  
*Function to read data from the other endpoint.*
- bool [sock\\_send\\_food](#) (int socket, char \*data)  
*Higher level function to send serialized food to the other endpoint.*
- bool [sock\\_send\\_search](#) (int socket, char \*data)  
*Higher level function to send a search request to the other endpoint.*
- bool [sock\\_send\\_count](#) (int socket, char \*data)  
*Higher level function to send the number of found items to the other endpoint.*

## 4.10.1 Detailed Description

Header file containing read and write functions for calory socket protocol.

## Author

Lukas Elsner

## Date

02-09-2014 Every write is BUF\_LEN bytes long and has to be acknowledged with a RE\_LEN bytes long answer containing ACK or NACK.

## 4.10.2 Function Documentation

## 4.10.2.1 size\_t sock\_read ( int socket, char \* data )

Function to read data from the other endpoint.

**Parameters**

<i>int</i>	The socket to communicate with
<i>char*</i>	A pointer to a buffer which for the read data. Must be at least 4096 bytes long.

**Returns**

The size of read data

**4.10.2.2 bool sock\_send\_count ( int *socket*, char \* *data* )**

Higher level function to send the number of found items to the other endpoint.

**Parameters**

<i>int</i>	The socket to communicate with
<i>char*</i>	The number of found items as string value

**Returns**

True, if the communication was successful, false otherwise

**4.10.2.3 bool sock\_send\_food ( int *socket*, char \* *data* )**

Higher level function to send serialized food to the other endpoint.

**Parameters**

<i>int</i>	The socket to communicate with
<i>char*</i>	The serialized food to send

**Returns**

True, if the communication was successful, false otherwise

**4.10.2.4 bool sock\_send\_search ( int *socket*, char \* *data* )**

Higher level function to send a search request to the other endpoint.

**Parameters**

<i>int</i>	The socket to communicate with
<i>char*</i>	The search term to send

**Returns**

True, if the communication was successful, false otherwise

**4.10.2.5 bool sock\_write ( int *socket*, char \* *data* )**

Lower level function to send data to the other endpoint.

**Parameters**

<i>int</i>	The socket to communicate with
<i>char*</i>	The data to send

**Returns**

True, if the communication was successful, false otherwise

## 4.11 sockethandler.c File Reference

File containing the sockethandler structure and its member methods.

```
#include <time.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <stdbool.h>
#include <sys/select.h>
#include <semaphore.h>
#include <pthread.h>
#include <assert.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include "../lib/sock.h"
#include "../lib/food.h"
#include "../lib/foodlist.h"
#include "sockethandler.h"
```

### Data Structures

- struct [sockethandler](#)  
*sockethandler structure for representing a sockethandler item*

### Macros

- #define [MAX\\_THREADS](#) 10
- #define [MAX\\_SOCKETS](#) 5

### Functions

- void [sockethandler\\_client\\_thread\\_func](#) ([sockethandler](#) \*s)  
*Method for client connection handling.*
- [sockethandler](#) \* [sockethandler\\_init](#) ([foodlist](#) \*fl)  
*Constructor for sockethandler.*
- void [sockethandler\\_server\\_thread\\_func](#) ([sockethandler](#) \*s)  
*Main loop function for the sockethandling procedure.*
- void [sockethandler\\_set\\_port](#) ([sockethandler](#) \*s, int port)  
*Method for setting the listening port of a sockethandler structure.*
- void [sockethandler\\_shutdown](#) ([sockethandler](#) \*s)  
*Function to notify main loop thread, that it should shut down.*
- void [sockethandler\\_destroy](#) ([sockethandler](#) \*s)  
*Destructor for sockethandler.*

#### 4.11.1 Detailed Description

File containing the sockethandler structure and its member methods.

### Author

Lukas Elsner

## Date

01-09-2014

## 4.11.2 Macro Definition Documentation

4.11.2.1 `#define MAX_SOCKETS 5`

Maximum number of waiting clients

4.11.2.2 `#define MAX_THREADS 10`

Size of the Threadpool

## 4.11.3 Function Documentation

4.11.3.1 `void sockethandler_client_thread_func ( sockethandler * s )`

Method for client connection handling.

## Parameters

<i>sockethandler*</i>	A pointer to a valid sockethandler structure
-----------------------	--

Every Thread is a consumer for the `client_sockets[]` array. If a socket is available, it is popped out by one of the threads and served in a loop until the connection closes. After that, the thread waits for its next client socket.

4.11.3.2 `void sockethandler_destroy ( sockethandler * )`

Destructor for sockethandler.

## Parameters

<i>sockethandler*</i>	Pointer to structure to be freed
-----------------------	----------------------------------

4.11.3.3 `sockethandler* sockethandler_init ( foodlist * )`

Constructor for sockethandler.

## Returns

A pointer to the sockethandler structure, representing the created object

After using this structure, it must be freed with `sockethandler_destroy(food *)`4.11.3.4 `void sockethandler_server_thread_func ( sockethandler * s )`

Main loop function for the sockethandling procedure.

## Parameters

<i>sockethandler*</i>	A pointer to a valid initialized sockethandler structure
-----------------------	--

This method starts a listening socket and produces client sockets for the spawned threads which are responsible for client connection handling. The method returns after `sockethandler_shutdown()` was called and all threads ended gracefully.

4.11.3.5 `void sockethandler_set_port ( sockethandler * s, int port )`

Method for setting the listening port of a sockethandler structure.

## Parameters

<i>sockethandler*</i>	Pointer to structure to work on
<i>int</i>	Port to listen on

## 4.11.3.6 void sockethandler\_shutdown ( sockethandler \* s )

Function to notify main loop thread, that it should shut down.

## Parameters

<i>sockethandler*</i>	A pointer to a valid initialized sockethandler structure
-----------------------	--

This method sets the shutdown flag for the sockethandler structure. After that it joins all running threads before it returns.

## 4.12 sockethandler.h File Reference

Header containing the public accessible sockethandler methods.

```
#include "../lib/foodlist.h"
```

## Typedefs

- typedef struct [sockethandler](#) [sockethandler](#)  
*Forward declaration for food.*

## Functions

- [sockethandler \\*](#) [sockethandler\\_init](#) ([foodlist \\*](#))  
*Constructor for sockethandler.*
- void [sockethandler\\_server\\_thread\\_func](#) ([sockethandler \\*](#)s)  
*Main loop function for the sockethandling procedure.*
- void [sockethandler\\_shutdown](#) ([sockethandler \\*](#)s)  
*Function to notify main loop thread, that it should shut down.*
- void [sockethandler\\_set\\_port](#) ([sockethandler \\*](#)s, int port)  
*Method for setting the listening port of a sockethandler structure.*
- void [sockethandler\\_destroy](#) ([sockethandler \\*](#))  
*Destructor for sockethandler.*

## 4.12.1 Detailed Description

Header containing the public accessible sockethandler methods.

## Author

Lukas Elsner

## Date

25-09-2014

## 4.12.2 Function Documentation

## 4.12.2.1 void sockethandler\_destroy ( sockethandler \* )

Destructor for sockethandler.

**Parameters**

<i>sockethandler*</i>	Pointer to structure to be freed
-----------------------	----------------------------------

**4.12.2.2 sockethandler\* sockethandler\_init ( foodlist \* )**

Constructor for sockethandler.

**Returns**

A pointer to the sockethandler structure, representing the created object

After using this structure, it must be freed with `sockethandler_destroy(food *)`

**4.12.2.3 void sockethandler\_server\_thread\_func ( sockethandler \* s )**

Main loop function for the sockethandling procedure.

**Parameters**

<i>sockethandler*</i>	A pointer to a valid initialized sockethandler structure
-----------------------	--

This method starts a listening socket and produces client sockets for the spawned threads which are responsible for client connection handling. The method returns after `sockethandler_shutdown()` was called and all threads ended gracefully.

**4.12.2.4 void sockethandler\_set\_port ( sockethandler \* s, int port )**

Method for setting the listening port of a sockethandler structure.

**Parameters**

<i>sockethandler*</i>	Pointer to structure to work on
<i>int</i>	Port to listen on

**4.12.2.5 void sockethandler\_shutdown ( sockethandler \* s )**

Function to notify main loop thread, that it should shut down.

**Parameters**

<i>sockethandler*</i>	A pointer to a valid initialized sockethandler structure
-----------------------	--

This method sets the shutdown flag for the sockethandler structure. After that it joins all running threads before it returns.



## Index

- ask\_user\_cancel
  - diet-client.c, [7](#)
- carbo
  - food, [3](#)
- client\_config, [2](#)
  - host, [3](#)
  - port, [3](#)
- client\_loop
  - diet-client.c, [7](#)
- client\_sockets
  - sockethandler, [5](#)
- cmpfunc
  - foodlist.c, [21](#)
- count
  - sockethandler, [5](#)
- data
  - foodlist, [4](#)
- diet-client.c, [6](#)
  - ask\_user\_cancel, [7](#)
  - client\_loop, [7](#)
  - get\_food\_from\_user, [8](#)
  - main, [8](#)
  - usage, [8](#)
- diet-server.c, [8](#)
  - main, [9](#)
  - usage, [9](#)
- empty
  - sockethandler, [5](#)
- end\_read
  - foodlist.c, [21](#)
- end\_write
  - foodlist.c, [22](#)
- fat
  - food, [3](#)
- file
  - foodlist, [4](#)
- food, [3](#)
  - carbo, [3](#)
  - fat, [3](#)
  - kcal, [3](#)
  - measure, [3](#)
  - name, [3](#)
  - protein, [3](#)
  - weight, [3](#)
- food.c, [9](#)
  - food\_deserialize, [11](#)
  - food\_destroy, [11](#)
  - food\_get\_carbo, [11](#)
  - food\_get\_fat, [11](#)
  - food\_get\_kcal, [11](#)
  - food\_get\_measure, [11](#)
  - food\_get\_name, [12](#)
  - food\_get\_protein, [12](#)
  - food\_get\_size, [12](#)
  - food\_get\_weight, [12](#)
  - food\_init, [12](#)
  - food\_serialize, [12](#)
  - food\_set\_carbo, [14](#)
  - food\_set\_fat, [14](#)
  - food\_set\_kcal, [14](#)
  - food\_set\_measure, [14](#)
  - food\_set\_name, [14](#)
  - food\_set\_protein, [14](#)
  - food\_set\_weight, [15](#)
  - food\_to\_string, [15](#)
- food.h, [15](#)
  - food\_deserialize, [16](#)
  - food\_destroy, [17](#)
  - food\_get\_carbo, [17](#)
  - food\_get\_fat, [17](#)
  - food\_get\_kcal, [17](#)
  - food\_get\_measure, [17](#)
  - food\_get\_name, [18](#)
  - food\_get\_protein, [18](#)
  - food\_get\_size, [18](#)
  - food\_get\_weight, [18](#)
  - food\_init, [18](#)
  - food\_serialize, [18](#)
  - food\_set\_carbo, [19](#)
  - food\_set\_fat, [19](#)
  - food\_set\_kcal, [19](#)
  - food\_set\_measure, [19](#)
  - food\_set\_name, [19](#)
  - food\_set\_protein, [19](#)
  - food\_set\_weight, [20](#)
  - food\_to\_string, [20](#)
- food\_deserialize
  - food.c, [11](#)
  - food.h, [16](#)
- food\_destroy
  - food.c, [11](#)
  - food.h, [17](#)
- food\_get\_carbo
  - food.c, [11](#)
  - food.h, [17](#)
- food\_get\_fat
  - food.c, [11](#)
  - food.h, [17](#)
- food\_get\_kcal
  - food.c, [11](#)
  - food.h, [17](#)
- food\_get\_measure
  - food.c, [11](#)
  - food.h, [17](#)
- food\_get\_name
  - food.c, [12](#)
  - food.h, [18](#)

- food\_get\_protein
  - food.c, [12](#)
  - food.h, [18](#)
- food\_get\_size
  - food.c, [12](#)
  - food.h, [18](#)
- food\_get\_weight
  - food.c, [12](#)
  - food.h, [18](#)
- food\_init
  - food.c, [12](#)
  - food.h, [18](#)
- food\_serialize
  - food.c, [12](#)
  - food.h, [18](#)
- food\_set\_carbo
  - food.c, [14](#)
  - food.h, [19](#)
- food\_set\_fat
  - food.c, [14](#)
  - food.h, [19](#)
- food\_set\_kcal
  - food.c, [14](#)
  - food.h, [19](#)
- food\_set\_measure
  - food.c, [14](#)
  - food.h, [19](#)
- food\_set\_name
  - food.c, [14](#)
  - food.h, [19](#)
- food\_set\_protein
  - food.c, [14](#)
  - food.h, [19](#)
- food\_set\_weight
  - food.c, [15](#)
  - food.h, [20](#)
- food\_to\_string
  - food.c, [15](#)
  - food.h, [20](#)
- foodlist, [4](#)
  - data, [4](#)
  - file, [4](#)
  - r\_mutex, [4](#)
  - read\_count, [4](#)
  - rw\_mutex, [4](#)
  - sockethandler, [6](#)
- foodlist.c, [20](#)
  - cmpfunc, [21](#)
  - end\_read, [21](#)
  - end\_write, [22](#)
  - foodlist\_append, [22](#)
  - foodlist\_count, [22](#)
  - foodlist\_destroy, [22](#)
  - foodlist\_find, [22](#)
  - foodlist\_get\_data, [22](#)
  - foodlist\_init, [23](#)
  - foodlist\_init\_csv, [23](#)
  - foodlist\_is\_empty, [23](#)
- foodlist.h, [24](#)
  - foodlist\_save, [23](#)
  - start\_read, [23](#)
  - start\_write, [24](#)
- foodlistnode, [4](#)
  - item, [5](#)
  - next, [5](#)
- foodlistnode.c, [26](#)
  - foodlistnode\_count, [27](#)
  - foodlistnode\_destroy, [27](#)
  - foodlistnode\_get\_item, [28](#)
  - foodlistnode\_get\_next, [28](#)
  - foodlistnode\_has\_next, [28](#)
  - foodlistnode\_init, [28](#)
  - foodlistnode\_set\_item, [28](#)
  - foodlistnode\_set\_next, [29](#)
- foodlistnode.h, [29](#)
  - foodlistnode\_count, [30](#)
  - foodlistnode\_destroy, [30](#)
  - foodlistnode\_get\_item, [30](#)
  - foodlistnode\_get\_next, [30](#)
  - foodlistnode\_has\_next, [30](#)

- foodlistnode\_init, 30
- foodlistnode\_set\_item, 31
- foodlistnode\_set\_next, 31
- foodlistnode\_count
  - foodlistnode.c, 27
  - foodlistnode.h, 30
- foodlistnode\_destroy
  - foodlistnode.c, 27
  - foodlistnode.h, 30
- foodlistnode\_get\_item
  - foodlistnode.c, 28
  - foodlistnode.h, 30
- foodlistnode\_get\_next
  - foodlistnode.c, 28
  - foodlistnode.h, 30
- foodlistnode\_has\_next
  - foodlistnode.c, 28
  - foodlistnode.h, 30
- foodlistnode\_init
  - foodlistnode.c, 28
  - foodlistnode.h, 30
- foodlistnode\_set\_item
  - foodlistnode.c, 28
  - foodlistnode.h, 31
- foodlistnode\_set\_next
  - foodlistnode.c, 29
  - foodlistnode.h, 31
- full
  - sockethandler, 6
- get\_food\_from\_user
  - diet-client.c, 8
- host
  - client\_config, 3
- in
  - sockethandler, 6
- item
  - foodlistnode, 5
- kcal
  - food, 3
- listen\_port
  - sockethandler, 6
- MAX\_SOCKETS
  - sockethandler.c, 36
- MAX\_THREADS
  - sockethandler.c, 36
- main
  - diet-client.c, 8
  - diet-server.c, 9
- measure
  - food, 3
- mutex
  - sockethandler, 6
- name
  - food, 3
- next
  - foodlistnode, 5
- out
  - sockethandler, 6
- port
  - client\_config, 3
- protein
  - food, 3
- r\_mutex
  - foodlist, 4
- read\_count
  - foodlist, 4
- rw\_mutex
  - foodlist, 4
- shutdown
  - sockethandler, 6
- sock.c, 31
  - sock\_read, 32
  - sock\_send\_count, 32
  - sock\_send\_food, 32
  - sock\_send\_search, 32
  - sock\_write, 32
- sock.h, 33
  - sock\_read, 33
  - sock\_send\_count, 34
  - sock\_send\_food, 34
  - sock\_send\_search, 34
  - sock\_write, 34
- sock\_read
  - sock.c, 32
  - sock.h, 33
- sock\_send\_count
  - sock.c, 32
  - sock.h, 34
- sock\_send\_food
  - sock.c, 32
  - sock.h, 34
- sock\_send\_search
  - sock.c, 32
  - sock.h, 34
- sock\_write
  - sock.c, 32
  - sock.h, 34
- sockethandler, 5
  - client\_sockets, 5
  - count, 5
  - empty, 5
  - foodlist, 6
  - full, 6
  - in, 6
  - listen\_port, 6
  - mutex, 6
  - out, 6
  - shutdown, 6

- thread\_pool, 6
- sockethandler.c, 35
  - MAX\_SOCKETS, 36
  - MAX\_THREADS, 36
  - sockethandler\_client\_thread\_func, 36
  - sockethandler\_destroy, 36
  - sockethandler\_init, 36
  - sockethandler\_server\_thread\_func, 36
  - sockethandler\_set\_port, 36
  - sockethandler\_shutdown, 37
- sockethandler.h, 37
  - sockethandler\_destroy, 37
  - sockethandler\_init, 38
  - sockethandler\_server\_thread\_func, 38
  - sockethandler\_set\_port, 38
  - sockethandler\_shutdown, 38
- sockethandler\_client\_thread\_func
  - sockethandler.c, 36
- sockethandler\_destroy
  - sockethandler.c, 36
  - sockethandler.h, 37
- sockethandler\_init
  - sockethandler.c, 36
  - sockethandler.h, 38
- sockethandler\_server\_thread\_func
  - sockethandler.c, 36
  - sockethandler.h, 38
- sockethandler\_set\_port
  - sockethandler.c, 36
  - sockethandler.h, 38
- sockethandler\_shutdown
  - sockethandler.c, 37
  - sockethandler.h, 38
- start\_read
  - foodlist.c, 23
- start\_write
  - foodlist.c, 24
- thread\_pool
  - sockethandler, 6
- usage
  - diet-client.c, 8
  - diet-server.c, 9
- weight
  - food, 3