

ZipZop

Generated by Doxygen 1.8.13

Contents

1	Data Structure Index	1
1.1	Data Structures	1
2	File Index	3
2.1	File List	3
3	Data Structure Documentation	5
3.1	client Struct Reference	5
3.1.1	Detailed Description	5
3.1.2	Field Documentation	5
3.1.2.1	name	5
3.1.2.2	sockfd	5
3.1.2.3	thread	6
3.2	message Struct Reference	6
3.2.1	Detailed Description	6
3.2.2	Field Documentation	6
3.2.2.1	content	6
3.2.2.2	sender_name	6
3.3	sllist Struct Reference	7
3.3.1	Detailed Description	7
3.3.2	Field Documentation	7
3.3.2.1	key	7
3.3.2.2	next	7

4 File Documentation	9
4.1 src/client.c File Reference	9
4.1.1 Function Documentation	10
4.1.1.1 client_create()	10
4.1.1.2 client_destroy()	10
4.1.1.3 client_get_name()	11
4.1.1.4 client_get_socket()	11
4.1.1.5 client_get_thread()	11
4.1.1.6 client_set_name()	12
4.1.1.7 client_set_socket()	12
4.1.1.8 client_set_thread()	12
4.2 src/client.h File Reference	13
4.2.1 Function Documentation	14
4.2.1.1 client_create()	14
4.2.1.2 client_destroy()	14
4.2.1.3 client_get_name()	15
4.2.1.4 client_get_socket()	15
4.2.1.5 client_get_thread()	15
4.2.1.6 client_set_name()	16
4.2.1.7 client_set_socket()	16
4.2.1.8 client_set_thread()	16
4.3 src/errcodes.h File Reference	17
4.3.1 Enumeration Type Documentation	17
4.3.1.1 errcodes	17
4.4 src/message.c File Reference	18
4.4.1 Function Documentation	18
4.4.1.1 message_create()	19
4.4.1.2 message_destroy()	19
4.4.1.3 message_get_content()	19
4.4.1.4 message_get_sender()	20

4.4.1.5	message_pack()	20
4.4.1.6	message_unpack()	21
4.5	src/message.h File Reference	21
4.5.1	Function Documentation	22
4.5.1.1	message_create()	22
4.5.1.2	message_destroy()	23
4.5.1.3	message_get_content()	23
4.5.1.4	message_get_sender()	24
4.5.1.5	message_pack()	24
4.5.1.6	message_unpack()	25
4.6	src/sllist.c File Reference	25
4.6.1	Function Documentation	26
4.6.1.1	sll_get_key()	26
4.6.1.2	sll_get_next()	26
4.6.1.3	sll_init()	27
4.6.1.4	sll_insert_first()	27
4.6.1.5	sll_insert_last()	28
4.6.1.6	sll_remove_elm()	28
4.6.1.7	sll_remove_first()	28
4.6.1.8	sll_remove_last()	29
4.7	src/sllist.h File Reference	29
4.7.1	Macro Definition Documentation	30
4.7.1.1	SLL_INIT	31
4.7.2	Function Documentation	31
4.7.2.1	sll_get_key()	31
4.7.2.2	sll_get_next()	31
4.7.2.3	sll_init()	32
4.7.2.4	sll_insert_first()	32
4.7.2.5	sll_insert_last()	33
4.7.2.6	sll_remove_elm()	33

4.7.2.7	<code>sll_remove_first()</code>	33
4.7.2.8	<code>sll_remove_last()</code>	34
4.8	<code>src/zip-zop-client.c</code> File Reference	34
4.8.1	Macro Definition Documentation	35
4.8.1.1	<code>MESSAGE_LEN</code>	35
4.8.1.2	<code>PORT</code>	35
4.8.2	Function Documentation	35
4.8.2.1	<code>check_args()</code>	35
4.8.2.2	<code>communicate()</code>	36
4.8.2.3	<code>create_and_connect()</code>	36
4.8.2.4	<code>get_server_addr()</code>	36
4.8.2.5	<code>listen_thread()</code>	36
4.8.2.6	<code>main()</code>	36
4.8.2.7	<code>print_usage()</code>	36
4.8.2.8	<code>server_introduction()</code>	36
4.8.2.9	<code>show_message()</code>	37
4.8.2.10	<code>speak_thread()</code>	37
4.9	<code>src/zip-zop-server.c</code> File Reference	37
4.9.1	Macro Definition Documentation	38
4.9.1.1	<code>BACKLOG</code>	38
4.9.1.2	<code>CLIENT_NAME_LEN</code>	38
4.9.1.3	<code>MESSAGE_LEN</code>	39
4.9.1.4	<code>PORT</code>	39
4.9.2	Function Documentation	39
4.9.2.1	<code>accept_clients()</code>	39
4.9.2.2	<code>client_thread_broadcast()</code>	39
4.9.2.3	<code>client_thread_listen()</code>	40
4.9.2.4	<code>create_and_bind()</code>	40
4.9.2.5	<code>create_new_client()</code>	40
4.9.2.6	<code>get_internet_addr()</code>	41
4.9.2.7	<code>kill_client()</code>	41
4.9.2.8	<code>main()</code>	41
4.9.3	Variable Documentation	42
4.9.3.1	<code>CLIENT_LIST</code>	42
4.9.3.2	<code>CLIENT_LIST_MUTEX</code>	42

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

client	Struct representing a connect client in the server	5
message	Struct representing a messege sent by some sender	6
sllist	A struct representing node in a singly linked list	7

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

src/ client.c	9
src/ client.h	13
src/ errcodes.h	17
src/ message.c	18
src/ message.h	21
src/ slist.c	25
src/ slist.h	29
src/ zip-zop-client.c	34
src/ zip-zop-server.c	37

Chapter 3

Data Structure Documentation

3.1 client Struct Reference

Struct representing a connect client in the server.

Data Fields

- const char * [name](#)
- int [sockfd](#)
- pthread_t [thread](#)

3.1.1 Detailed Description

Struct representing a connect client in the server.

3.1.2 Field Documentation

3.1.2.1 name

```
const char* client::name
```

Client name

3.1.2.2 sockfd

```
int client::sockfd
```

Socket that holds the connection with this client

3.1.2.3 thread

```
pthread_t client::thread
```

The server thread responsible to listen to this client's messages

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

- [src/client.c](#)

3.2 message Struct Reference

Struct representing a message sent by some sender.

Data Fields

- const char * [content](#)
- const char * [sender_name](#)

3.2.1 Detailed Description

Struct representing a message sent by some sender.

3.2.2 Field Documentation

3.2.2.1 content

```
const char* message::content
```

The content of the message

3.2.2.2 sender_name

```
const char* message::sender_name
```

The username of the sender

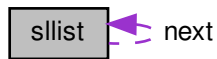
The documentation for this struct was generated from the following file:

- [src/message.c](#)

3.3 slist Struct Reference

A struct representing node in a singly linked list.

Collaboration diagram for slist:



Data Fields

- void * [key](#)
- struct [slist](#) * [next](#)

3.3.1 Detailed Description

A struct representing node in a singly linked list.

3.3.2 Field Documentation

3.3.2.1 key

```
void* slist::key
```

The element that will be stored in the node

3.3.2.2 next

```
struct slist* slist::next
```

A pointer to the next node

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

- [src/slist.c](#)

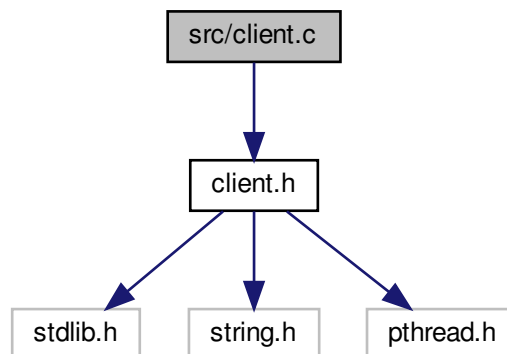
Chapter 4

File Documentation

4.1 src/client.c File Reference

```
#include "client.h"
```

Include dependency graph for client.c:



Data Structures

- struct `client`

Struct representing a connect client in the server.

Functions

- struct `client` * `client_create` (const char *name, int sockfd)
Create a client instance.
- void `client_destroy` (struct `client` *c)
- const char * `client_get_name` (struct `client` *c)

- Get the client name.*
- int [client_get_socket](#) (struct [client](#) *c)
- Get the client socket.*
- pthread_t * [client_get_thread](#) (struct [client](#) *c)
- Get the client thread.*
- void [client_set_name](#) (struct [client](#) *c, const char *name)
- Set the client name.*
- void [client_set_socket](#) (struct [client](#) *c, int sockfd)
- Set the client socket.*
- void [client_set_thread](#) (struct [client](#) *c, pthread_t thread)
- Set the client thread.*

4.1.1 Function Documentation

4.1.1.1 [client_create\(\)](#)

```
struct client* client_create (
    const char * name,
    int sockfd )
```

Create a client instance.

Both parameters will be copied into the message, so the user is free to `free()` the parameters passed to this function if necessary.

Parameters

in	<i>name</i>	The client name.
in	<i>sockfd</i>	The socket connected to this client.

Returns

A pointer to the client in case of success, NULL otherwise. The client must be freed, using [client_destroy\(\)](#).

See also

[client_destroy](#)

4.1.1.2 [client_destroy\(\)](#)

```
void client_destroy (
    struct client * c )
```

Destroys a client.

Parameters

in	<i>c</i>	A pointer to the client.
----	----------	--------------------------

4.1.1.3 client_get_name()

```
const char* client_get_name (  
    struct client * c )
```

Get the client name.

Parameters

in	<i>c</i>	The client.
----	----------	-------------

Returns

The client name.

4.1.1.4 client_get_socket()

```
int client_get_socket (  
    struct client * c )
```

Get the client socket.

Parameters

in	<i>c</i>	The client.
----	----------	-------------

Returns

The client socket.

4.1.1.5 client_get_thread()

```
pthread_t* client_get_thread (  
    struct client * c )
```

Get the client thread.

Parameters

in	<i>c</i>	The client.
----	----------	-------------

Returns

An Address of the client thread.

Warning

This function returns the address of the actual thread stored in the client. Do not try to free this address.

4.1.1.6 client_set_name()

```
void client_set_name (
    struct client * c,
    const char * name )
```

Set the client name.

Parameters

in	<i>c</i>	The client.
in	<i>name</i>	The client name.

4.1.1.7 client_set_socket()

```
void client_set_socket (
    struct client * c,
    int sockfd )
```

Set the client socket.

Parameters

in	<i>c</i>	The client.
in	<i>sockfd</i>	The client socket.

4.1.1.8 client_set_thread()

```
void client_set_thread (
    struct client * c,
    pthread_t thread )
```

Set the client thread.

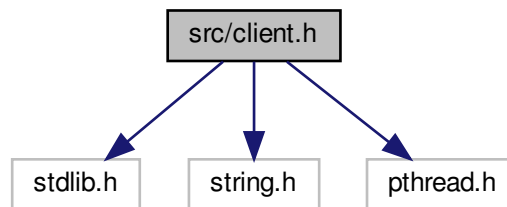
Parameters

in	<i>c</i>	The client.
in	<i>thread</i>	The client thread.

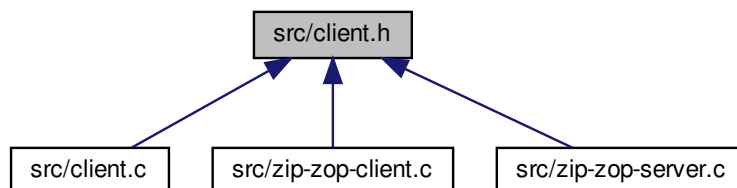
4.2 src/client.h File Reference

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

Include dependency graph for client.h:



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



Functions

- struct [client](#) * [client_create](#) (const char *name, int sockfd)
Create a client instance.
- void [client_destroy](#) (struct [client](#) *c)
- const char * [client_get_name](#) (struct [client](#) *c)
Get the client name.

- `int client_get_socket (struct client *c)`
Get the client socket.
- `pthread_t * client_get_thread (struct client *c)`
Get the client thread.
- `void client_set_name (struct client *c, const char *name)`
Set the client name.
- `void client_set_socket (struct client *c, int sockfd)`
Set the client socket.
- `void client_set_thread (struct client *c, pthread_t thread)`
Set the client thread.

4.2.1 Function Documentation

4.2.1.1 client_create()

```
struct client* client_create (
    const char * name,
    int sockfd )
```

Create a client instance.

Both parameters will be copied into the message, so the user is free to `free()` the parameters passed to this function if necessary.

Parameters

in	<i>name</i>	The client name.
in	<i>sockfd</i>	The socket connected to this client.

Returns

A pointer to the client in case of success, NULL otherwise. The client must be freed, using [client_destroy\(\)](#).

See also

[client_destroy](#)

4.2.1.2 client_destroy()

```
void client_destroy (
    struct client * c )
```

Destroys a client.

Parameters

in	<i>c</i>	A pointer to the client.
----	----------	--------------------------

4.2.1.3 client_get_name()

```
const char* client_get_name (  
    struct client * c )
```

Get the client name.

Parameters

in	<i>c</i>	The client.
----	----------	-------------

Returns

The client name.

4.2.1.4 client_get_socket()

```
int client_get_socket (  
    struct client * c )
```

Get the client socket.

Parameters

in	<i>c</i>	The client.
----	----------	-------------

Returns

The client socket.

4.2.1.5 client_get_thread()

```
pthread_t* client_get_thread (  
    struct client * c )
```

Get the client thread.

Parameters

in	<i>c</i>	The client.
----	----------	-------------

Returns

An Address of the client thread.

Warning

This function returns the address of the actual thread stored in the client. Do not try to free this address.

4.2.1.6 client_set_name()

```
void client_set_name (
    struct client * c,
    const char * name )
```

Set the client name.

Parameters

in	<i>c</i>	The client.
in	<i>name</i>	The client name.

4.2.1.7 client_set_socket()

```
void client_set_socket (
    struct client * c,
    int sockfd )
```

Set the client socket.

Parameters

in	<i>c</i>	The client.
in	<i>sockfd</i>	The client socket.

4.2.1.8 client_set_thread()

```
void client_set_thread (
    struct client * c,
    pthread_t thread )
```

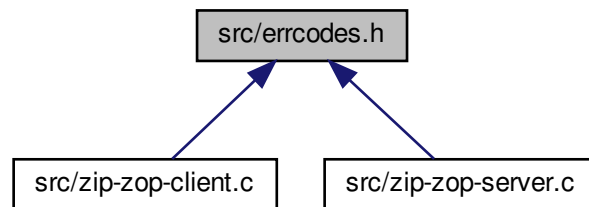
Set the client thread.

Parameters

in	<i>c</i>	The client.
in	<i>thread</i>	The client thread.

4.3 src/errcodes.h File Reference

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



Enumerations

- enum [errcodes](#) {
[E_SUCCESS](#), [E_GETADDRINFO](#), [E_BIND](#), [E_LISTEN](#),
[E_BAD_ARGS](#), [E_CONNECT](#), [E_PTHREAD_CREATE](#) }

Possible error codes in the project.

4.3.1 Enumeration Type Documentation

4.3.1.1 errcodes

enum [errcodes](#)

Possible error codes in the project.

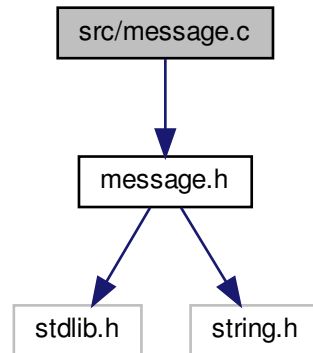
Enumerator

E_SUCCESS	Success value
E_GETADDRINFO	Error code if getaddrinfo() fails
E_BIND	Error code if it was not possible to bind() in the specified port
E_LISTEN	Error code if liste() fails
E_BAD_ARGS	Error code if the user gave a bad input
E_CONNECT	Error code if connect() fails
E_PTHREAD_CREATE	Error code if it was not possible to create a new thread

4.4 src/message.c File Reference

```
#include "message.h"
```

Include dependency graph for message.c:



Data Structures

- struct [message](#)
Struct representing a message sent by some sender.

Functions

- struct [message](#) * [message_create](#) (const char *content, const char *sender_name)
Creates a message.
- void [message_destroy](#) (struct [message](#) *m)
Destroys a message.
- const char * [message_get_content](#) (struct [message](#) *m)
Get the message content.
- const char * [message_get_sender](#) (struct [message](#) *m)
Get the message sender.
- char * [message_pack](#) (struct [message](#) *m, int *len)
Serialize a message.
- struct [message](#) * [message_unpack](#) (char *pack)

4.4.1 Function Documentation

4.4.1.1 message_create()

```
struct message* message_create (
    const char * content,
    const char * sender_name )
```

Creates a message.

Both parameters will be copied into the message, so the user is free to `free()` the parameters passed to this function if necessary.

Parameters

in	<i>content</i>	The content of the message.
in	<i>sender_name</i>	The username of the sender.

Returns

A pointer to a struct message in case of success, NULL otherwise. The message must be freed, using [message_destroy\(\)](#), when is not needed anymore.

See also

[message_destroy](#)

4.4.1.2 message_destroy()

```
void message_destroy (
    struct message * m )
```

Destroys a message.

Parameters

in	<i>m</i>	A pointer to the message.
----	----------	---------------------------

See also

[message_create](#)

4.4.1.3 message_get_content()

```
const char* message_get_content (
    struct message * m )
```

Get the message content.

Parameters

in	<i>m</i>	A pointer to the message.
----	----------	---------------------------

Returns

A pointer to the message content.

Warning

The returned value should not be freed.

4.4.1.4 message_get_sender()

```
const char* message_get_sender (  
    struct message * m )
```

Get the message sender.

Parameters

in	<i>m</i>	A pointer to the message.
----	----------	---------------------------

Returns

A pointer to the sender name.

Warning

The returned value should not be freed.

4.4.1.5 message_pack()

```
char* message_pack (  
    struct message * m,  
    int * len )
```

Serialize a message.

Pack/Serialize the struct message in a format that can be sent through the network.

Parameters

in	<i>m</i>	A pointer to the message.
out	<i>len</i>	A pointer to a integer where the length of the serialized message will be stored.

Returns

A pointer to the serialized message. This should be freed when is not necessary anymore.

See also

[message_unpack](#)

4.4.1.6 message_unpack()

```
struct message* message_unpack (  
    char * pack )
```

Deserialize a message.

Unpack/Deserialize a string into a struct message.

Parameters

in	<i>pack</i>	The string that represent the packed message generated by message_pack() .
----	-------------	--------------------------------------------------------------------------------------------

Returns

A pointer to the deserialized message. This should be freed when is not necessary anymore.

See also

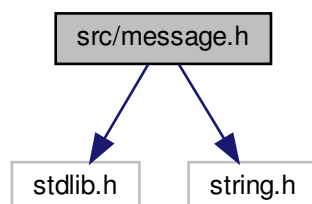
[message_pack](#)

4.5 src/message.h File Reference

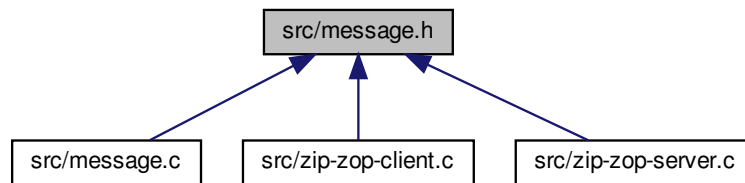
```
#include <stdlib.h>
```

```
#include <string.h>
```

Include dependency graph for message.h:



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



Functions

- struct [message](#) * [message_create](#) (const char *content, const char *sender_name)
Creates a message.
- void [message_destroy](#) (struct [message](#) *m)
Destroys a message.
- const char * [message_get_content](#) (struct [message](#) *m)
Get the message content.
- const char * [message_get_sender](#) (struct [message](#) *m)
Get the message sender.
- char * [message_pack](#) (struct [message](#) *m, int *len)
Serialize a message.
- struct [message](#) * [message_unpack](#) (char *pack)

4.5.1 Function Documentation

4.5.1.1 message_create()

```

struct message* message_create (
    const char * content,
    const char * sender_name )
  
```

Creates a message.

Both parameters will be copied into the message, so the user is free to `free()` the parameters passed to this function if necessary.

Parameters

in	<i>content</i>	The content of the message.
in	<i>sender_name</i>	The username of the sender.

Returns

A pointer to a struct message in case of success, NULL otherwise. The message must be freed, using [message_destroy\(\)](#), when is not needed anymore.

See also

[message_destroy](#)

4.5.1.2 message_destroy()

```
void message_destroy (  
    struct message * m )
```

Destroys a message.

Parameters

in	<i>m</i>	A pointer to the message.
----	----------	---------------------------

See also

[message_create](#)

4.5.1.3 message_get_content()

```
const char* message_get_content (  
    struct message * m )
```

Get the message content.

Parameters

in	<i>m</i>	A pointer to the message.
----	----------	---------------------------

Returns

A pointer to the message content.

Warning

The returned value should not be freed.

4.5.1.4 message_get_sender()

```
const char* message_get_sender (
    struct message * m )
```

Get the message sender.

Parameters

in	<i>m</i>	A pointer to the message.
----	----------	---------------------------

Returns

A pointer to the sender name.

Warning

The returned value should not be freed.

4.5.1.5 message_pack()

```
char* message_pack (
    struct message * m,
    int * len )
```

Serialize a message.

Pack/Serialize the struct message in a format that can be sent through the network.

Parameters

in	<i>m</i>	A pointer to the message.
out	<i>len</i>	A pointer to a integer where the length of the serialized message will be stored.

Returns

A pointer to the serialized message. This should be freed when is not necessary anymore.

See also

[message_unpack](#)

4.5.1.6 message_unpack()

```
struct message* message_unpack (
    char * pack )
```

Deserialize a message.

Unpack/Deserialize a string into a struct message.

Parameters

in	<i>pack</i>	The string that represent the packed message generated by message_pack() .
----	-------------	--------------------------------------------------------------------------------------------

Returns

A pointer to the deserialized message. This should be freed when is not necessary anymore.

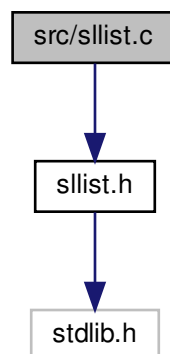
See also

[message_pack](#)

4.6 src/sllist.c File Reference

```
#include "sllist.h"
```

Include dependency graph for sllist.c:



Data Structures

- struct [sllist](#)

A struct representing node in a singly linked list.

Functions

- `struct sllist * sll_init (void)`
Initilize a sllist to be a valid empty list.
- `struct sllist * sll_get_next (struct sllist **l)`
Get the next node in the list.
- `void sll_insert_first (struct sllist **l, void *a)`
Insert an element on the head of the list.
- `void sll_insert_last (struct sllist **l, void *a)`
Insert an element on the tail of the list.
- `void * sll_remove_first (struct sllist **l)`
Remove the first element of the list.
- `void * sll_remove_last (struct sllist **l)`
Remove the last element of the list.
- `void * sll_remove_elm (struct sllist **l, void *elm)`
Remove the specified element of the list.
- `void * sll_get_key (struct sllist *l)`
Get the element stored in the especified list node.

4.6.1 Function Documentation

4.6.1.1 sll_get_key()

```
void* sll_get_key (
    struct sllist * l )
```

Get the element stored in the especified list node.

Parameters

in	/	A pointer to the list node.
----	---	-----------------------------

Returns

The element.

4.6.1.2 sll_get_next()

```
struct sllist* sll_get_next (
    struct sllist ** l )
```

Get the next node in the list.

Parameters

in, out	/	An address to a pointer to the list.
---------	---	--------------------------------------

Returns

A pointer to the next node in the list; NULL if there is no next element.

Example to iterate over a list:

```
struct sllist **l = sll_init();
// fill the list
for (struct sllist *p = *l; p; p = sll_get_next(&p)) {
    void *key = sll_get_key(p);
    // do stuff with p
}
```

4.6.1.3 sll_init()

```
struct sllist* sll_init (
    void )
```

Initilize a sllist to be a valid empty list.

Returns

An empty list.

Warning

One should not test the return against NULL. NULL is the default value.

See also

[SLL_INIT](#)

4.6.1.4 sll_insert_first()

```
void sll_insert_first (
    struct sllist ** l,
    void * a )
```

Insert an element on the head of the list.

Parameters

in, out	<i>l</i>	An address to a pointer to the list.
in	<i>a</i>	The element.

4.6.1.5 sll_insert_last()

```
void sll_insert_last (
    struct sllist ** l,
    void * a )
```

Insert an element on the tail of the list.

Parameters

in, out	<i>l</i>	An address to a pointer to the list.
in	<i>a</i>	The element.

4.6.1.6 sll_remove_elm()

```
void* sll_remove_elm (
    struct sllist ** l,
    void * elm )
```

Remove the specified element of the list.

Parameters

in, out	<i>l</i>	An address to a pointer to the list.
in	<i>elm</i>	The element.

Returns

The element in case of success. NULL if the list is empty or the element doesn't exit.

4.6.1.7 sll_remove_first()

```
void* sll_remove_first (
    struct sllist ** l )
```

Remove the first element of the list.

The list node will be freed.

Parameters

in, out	/	An address to a pointer to the list.
---------	---	--------------------------------------

Returns

The element in case of success. NULL if the list is empty.

4.6.1.8 sll_remove_last()

```
void* sll_remove_last (
    struct sllist ** l )
```

Remove the last element of the list.

The list node will be freed.

Parameters

in, out	/	An address to a pointer to the list.
---------	---	--------------------------------------

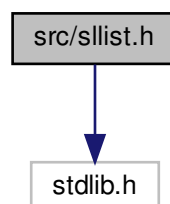
Returns

The element in case of success. NULL if the list is empty.

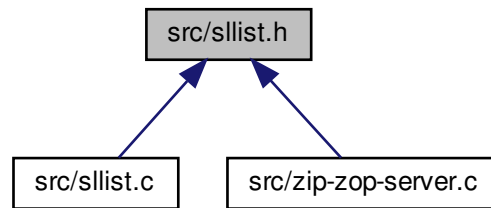
4.7 src/sllist.h File Reference

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

Include dependency graph for sllist.h:



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



Macros

- `#define SLL_INIT() NULL;`
Macro that initialize a slist to be a valid empty list.

Functions

- `struct slist * sll_init (void)`
Initilize a slist to be a valid empty list.
- `struct slist * sll_get_next (struct slist **l)`
Get the next node in the list.
- `void sll_insert_first (struct slist **l, void *a)`
Insert an element on the head of the list.
- `void sll_insert_last (struct slist **l, void *a)`
Insert an element on the tail of the list.
- `void * sll_remove_first (struct slist **l)`
Remove the first element of the list.
- `void * sll_remove_last (struct slist **l)`
Remove the last element of the list.
- `void * sll_remove_elm (struct slist **l, void *elm)`
Remove the specified element of the list.
- `void * sll_get_key (struct slist *l)`
Get the element stored in the especified list node.

4.7.1 Macro Definition Documentation

4.7.1.1 SLL_INIT

```
#define SLL_INIT( ) NULL;
```

Macro that initialize a slist to be a valid empty list.

Returns

An empty list.

Warning

One should not test the return against NULL. NULL is the default value.

See also

[sll_init](#)

4.7.2 Function Documentation

4.7.2.1 sll_get_key()

```
void* sll_get_key (
    struct sllist * l )
```

Get the element stored in the especified list node.

Parameters

in	/	A pointer to the list node.
----	---	-----------------------------

Returns

The element.

4.7.2.2 sll_get_next()

```
struct sllist* sll_get_next (
    struct sllist ** l )
```

Get the next node in the list.

Parameters

in, out	/	An address to a pointer to the list.
---------	---	--------------------------------------

Returns

A pointer to the next node in the list; NULL if there is no next element.

Example to iterate over a list:

```
struct sllist **l = sll_init();
// fill the list
for (struct sllist *p = *l; p; p = sll_get_next(&p)) {
    void *key = sll_get_key(p);
    // do stuff with p
}
```

4.7.2.3 sll_init()

```
struct sllist* sll_init (
    void )
```

Initilize a sllist to be a valid empty list.

Returns

An empty list.

Warning

One should not test the return against NULL. NULL is the default value.

See also

[SLL_INIT](#)

4.7.2.4 sll_insert_first()

```
void sll_insert_first (
    struct sllist ** l,
    void * a )
```

Insert an element on the head of the list.

Parameters

in, out	/	An address to a pointer to the list.
in	a	The element.

4.7.2.5 sll_insert_last()

```
void sll_insert_last (
    struct sllist ** l,
    void * a )
```

Insert an element on the tail of the list.

Parameters

in, out	/	An address to a pointer to the list.
in	a	The element.

4.7.2.6 sll_remove_elm()

```
void* sll_remove_elm (
    struct sllist ** l,
    void * elm )
```

Remove the specified element of the list.

Parameters

in, out	/	An address to a pointer to the list.
in	elm	The element.

Returns

The element in case of success. NULL if the list is empty or the element doesn't exit.

4.7.2.7 sll_remove_first()

```
void* sll_remove_first (
    struct sllist ** l )
```

Remove the first element of the list.

The list node will be freed.

Parameters

in, out	/	An address to a pointer to the list.
---------	---	--------------------------------------

Returns

The element in case of success. NULL if the list is empty.

4.7.2.8 sll_remove_last()

```
void* sll_remove_last (
    struct sllist ** l )
```

Remove the last element of the list.

The list node will be freed.

Parameters

in, out	/	An address to a pointer to the list.
---------	---	--------------------------------------

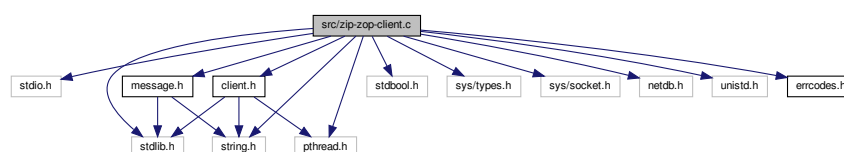
Returns

The element in case of success. NULL if the list is empty.

4.8 src/zip-zop-client.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <unistd.h>
#include <pthread.h>
#include "errcodes.h"
#include "message.h"
#include "client.h"
```

Include dependency graph for zip-zop-client.c:



Macros

- `#define PORT "1234"`
- `#define MESSAGE_LEN 2000`

Functions

- `bool check_args (int argc)`
- `void print_usage (const char *name)`
- `void show_message (struct message *m)`
- `void * listen_thread (void *client)`
- `void * speak_thread (void *client)`
- `struct addrinfo * get_server_addr (const char *server_name)`
- `int create_and_connect (struct addrinfo *addr)`
- `void server_introduction (struct client *c)`
- `void communicate (const char *user_name, int sockfd)`
- `int main (int argc, char **argv)`

4.8.1 Macro Definition Documentation

4.8.1.1 MESSAGE_LEN

```
#define MESSAGE_LEN 2000
```

4.8.1.2 PORT

```
#define PORT "1234"
```

4.8.2 Function Documentation

4.8.2.1 check_args()

```
bool check_args (  
    int argc )
```

4.8.2.2 communicate()

```
void communicate (
    const char * user_name,
    int sockfd )
```

4.8.2.3 create_and_connect()

```
int create_and_connect (
    struct addrinfo * addr )
```

4.8.2.4 get_server_addr()

```
struct addrinfo* get_server_addr (
    const char * server_name )
```

4.8.2.5 listen_thread()

```
void* listen_thread (
    void * client )
```

4.8.2.6 main()

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

4.8.2.7 print_usage()

```
void print_usage (
    const char * name )
```

4.8.2.8 server_introduction()

```
void server_introduction (
    struct client * c )
```

4.8.2.9 show_message()

```
void show_message (
    struct message * m )
```

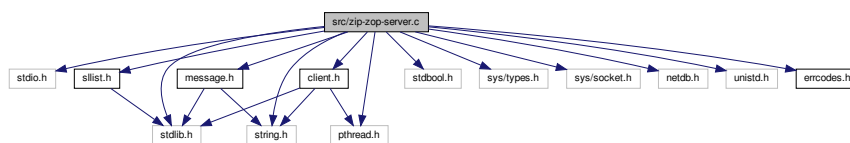
4.8.2.10 speak_thread()

```
void* speak_thread (
    void * client )
```

4.9 src/zip-zop-server.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <unistd.h>
#include <pthread.h>
#include "errcodes.h"
#include "message.h"
#include "client.h"
#include "sllist.h"
```

Include dependency graph for zip-zop-server.c:



Macros

- `#define PORT "1234"`
The port where this application will be running.
- `#define BACKLOG 10`
The number of clients that will be kept in the queue if the server is not ready for accepting them.
- `#define CLIENT_NAME_LEN 100`
Maximum length of a client name.
- `#define MESSAGE_LEN 2000`
Maximum length of a client message.

Functions

- void `client_thread_broadcast` (struct `client` *c, const char *msg)
Sends a message to all clients.
- void `kill_client` (struct `client` *c)
Kill a client.
- void * `client_thread_listen` (void *client)
Keeps listening to client messages.
- struct addrinfo * `get_internet_addr` (void)
Find a set of possible internet addresses of localhost.
- int `create_and_bind` (struct addrinfo *addr)
Attempts to create a socket and bind to a port with the given internet address.
- void `create_new_client` (int sockfd)
Create a new client and add it in the `CLIENT_LIST`.
- int `accept_clients` (int sockfd)
Keeps on accepting new clients connections.
- int `main` (void)
The zip-zop-server.

Variables

- struct `slist` * `CLIENT_LIST` = `SLL_INIT`()
A singly linked list that will keep all the connected clients.
- pthread_mutex_t `CLIENT_LIST_MUTEX`
The `CLIENT_LIST` mutex.

4.9.1 Macro Definition Documentation

4.9.1.1 BACKLOG

```
#define BACKLOG 10
```

The number of clients that will be kept in the queue if the server is not ready for accepting them.

4.9.1.2 CLIENT_NAME_LEN

```
#define CLIENT_NAME_LEN 100
```

Maximum length of a client name.

4.9.1.3 MESSAGE_LEN

```
#define MESSAGE_LEN 2000
```

Maximum length of a client message.

4.9.1.4 PORT

```
#define PORT "1234"
```

The port where this application will be running.

4.9.2 Function Documentation

4.9.2.1 accept_clients()

```
int accept_clients (
    int sockfd )
```

Keeps on accepting new clients connections.

Keeps listening for incoming connections, when a new one arrives accepts it and instantiates a new client.

Parameters

in	<i>sockfd</i>	Socket used to listen to new connections.
----	---------------	-------------------------------------------

4.9.2.2 client_thread_broadcast()

```
void client_thread_broadcast (
    struct client * c,
    const char * msg )
```

Sends a message to all clients.

The message will be sent as a packet version of a struct message.

Parameters

in	<i>c</i>	The client that sent the message.
in	<i>msg</i>	The message content.

See also

[message_pack](#)

4.9.2.3 client_thread_listen()

```
void* client_thread_listen (
    void * client )
```

Keeps listening to client messages.

This function will be executed by a thread that is responsible for keep checking if there is a new message from the client.

If there is an new message, the thread will execute the [client_thread_broadcast\(\)](#).

Parameters

in	<i>client</i>	A pointer to the client.
----	---------------	--------------------------

See also

[client_thread_broadcast](#)

4.9.2.4 create_and_bind()

```
int create_and_bind (
    struct addrinfo * addr )
```

Attempts to create a socket and bind to a port with the given internet address.

Parameters

in	<i>addr</i>	The internet address.
----	-------------	-----------------------

Returns

The socket in case of success. -1 otherwise.

4.9.2.5 create_new_client()

```
void create_new_client (
    int sockfd )
```

Create a new client and add it in the `CLIENT_LIST`.

Parameters

in	<i>sockfd</i>	The socket created in accept_clients() , and that is used to communicate with the client that will be created.
----	---------------	--------------------------------------------------------------------------------------------------------------------------------

See also

[accept_clients](#)
[CLIENT_LIST](#)

4.9.2.6 `get_internet_addr()`

```
struct addrinfo* get_internet_addr (
    void )
```

Find a set of possible internet addresses of localhost.

Returns

A list of `addrinfo`, wich contain the addresses.

4.9.2.7 `kill_client()`

```
void kill_client (
    struct client * c )
```

Kill a client.

Removes a client from the `CLIENT_LIST`, destroys it and closes the connection.

Parameters

in	<i>c</i>	The client.
----	----------	-------------

See also

[CLIENT_LIST](#)

4.9.2.8 `main()`

```
int main (
    void )
```

The zip-zop-server.

A TCP server that will accept connections from zip-zop-clients, hear its messages and broadcast them to all connected clients. Working as a chatroom.

4.9.3 Variable Documentation

4.9.3.1 CLIENT_LIST

```
struct sllist* CLIENT_LIST = SLL_INIT()
```

A singly linked list that will keep all the connected clients.

Warning

Mutual exclusion must be ensured before accessing this list.

See also

[CLIENT_LIST_MUTEX](#)

4.9.3.2 CLIENT_LIST_MUTEX

```
pthread_mutex_t CLIENT_LIST_MUTEX
```

The CLIENT_LIST mutex.

This is used to ensure mutual exclusion wen accessing the CLIENT_LIST, given the nature of the application where multiple threads might use the list.

See also

[CLIENT_LIST](#)

Index

- accept_clients
 - zip-zop-server.c, [39](#)
- BACKLOG
 - zip-zop-server.c, [38](#)
- CLIENT_LIST_MUTEX
 - zip-zop-server.c, [42](#)
- CLIENT_LIST
 - zip-zop-server.c, [42](#)
- CLIENT_NAME_LEN
 - zip-zop-server.c, [38](#)
- check_args
 - zip-zop-client.c, [35](#)
- client, [5](#)
 - name, [5](#)
 - sockfd, [5](#)
 - thread, [5](#)
- client.c
 - client_create, [10](#)
 - client_destroy, [10](#)
 - client_get_name, [11](#)
 - client_get_socket, [11](#)
 - client_get_thread, [11](#)
 - client_set_name, [12](#)
 - client_set_socket, [12](#)
 - client_set_thread, [12](#)
- client.h
 - client_create, [14](#)
 - client_destroy, [14](#)
 - client_get_name, [15](#)
 - client_get_socket, [15](#)
 - client_get_thread, [15](#)
 - client_set_name, [16](#)
 - client_set_socket, [16](#)
 - client_set_thread, [16](#)
- client_create
 - client.c, [10](#)
 - client.h, [14](#)
- client_destroy
 - client.c, [10](#)
 - client.h, [14](#)
- client_get_name
 - client.c, [11](#)
 - client.h, [15](#)
- client_get_socket
 - client.c, [11](#)
 - client.h, [15](#)
- client_get_thread
 - client.c, [11](#)
- client.h, [15](#)
 - client_set_name, [12](#)
 - client_set_socket, [12](#)
 - client_set_thread, [12](#)
- client_set_name
 - client.c, [12](#)
 - client.h, [16](#)
- client_set_socket
 - client.c, [12](#)
 - client.h, [16](#)
- client_set_thread
 - client.c, [12](#)
 - client.h, [16](#)
- client_thread_broadcast
 - zip-zop-server.c, [39](#)
- client_thread_listen
 - zip-zop-server.c, [40](#)
- communicate
 - zip-zop-client.c, [35](#)
- content
 - message, [6](#)
- create_and_bind
 - zip-zop-server.c, [40](#)
- create_and_connect
 - zip-zop-client.c, [36](#)
- create_new_client
 - zip-zop-server.c, [40](#)
- errcodes
 - errcodes.h, [17](#)
- errcodes.h
 - errcodes, [17](#)
- get_internet_addr
 - zip-zop-server.c, [41](#)
- get_server_addr
 - zip-zop-client.c, [36](#)
- key
 - sllist, [7](#)
- kill_client
 - zip-zop-server.c, [41](#)
- listen_thread
 - zip-zop-client.c, [36](#)
- MESSAGE_LEN
 - zip-zop-client.c, [35](#)
 - zip-zop-server.c, [38](#)
- main
 - zip-zop-client.c, [36](#)
 - zip-zop-server.c, [41](#)
- message, [6](#)
 - content, [6](#)

- sender_name, 6
- message.c
 - message_create, 18
 - message_destroy, 19
 - message_get_content, 19
 - message_get_sender, 20
 - message_pack, 20
 - message_unpack, 21
- message.h
 - message_create, 22
 - message_destroy, 23
 - message_get_content, 23
 - message_get_sender, 23
 - message_pack, 24
 - message_unpack, 24
- message_create
 - message.c, 18
 - message.h, 22
- message_destroy
 - message.c, 19
 - message.h, 23
- message_get_content
 - message.c, 19
 - message.h, 23
- message_get_sender
 - message.c, 20
 - message.h, 23
- message_pack
 - message.c, 20
 - message.h, 24
- message_unpack
 - message.c, 21
 - message.h, 24
- name
 - client, 5
- next
 - sllist, 7
- PORT
 - zip-zop-client.c, 35
 - zip-zop-server.c, 39
- print_usage
 - zip-zop-client.c, 36
- SLL_INIT
 - sllist.h, 30
- sender_name
 - message, 6
- server_introduction
 - zip-zop-client.c, 36
- show_message
 - zip-zop-client.c, 36
- sll_get_key
 - sllist.c, 26
 - sllist.h, 31
- sll_get_next
 - sllist.c, 26
 - sllist.h, 31
- sll_init
 - sllist.c, 27
 - sllist.h, 32
- sll_insert_first
 - sllist.c, 27
 - sllist.h, 32
- sll_insert_last
 - sllist.c, 28
 - sllist.h, 33
- sll_remove_elm
 - sllist.c, 28
 - sllist.h, 33
- sll_remove_first
 - sllist.c, 28
 - sllist.h, 33
- sll_remove_last
 - sllist.c, 29
 - sllist.h, 34
- sllist, 7
 - key, 7
 - next, 7
- sllist.c
 - sll_get_key, 26
 - sll_get_next, 26
 - sll_init, 27
 - sll_insert_first, 27
 - sll_insert_last, 28
 - sll_remove_elm, 28
 - sll_remove_first, 28
 - sll_remove_last, 29
- sllist.h
 - SLL_INIT, 30
 - sll_get_key, 31
 - sll_get_next, 31
 - sll_init, 32
 - sll_insert_first, 32
 - sll_insert_last, 33
 - sll_remove_elm, 33
 - sll_remove_first, 33
 - sll_remove_last, 34
- sockfd
 - client, 5
- speak_thread
 - zip-zop-client.c, 37
- src/client.c, 9
- src/client.h, 13
- src/errcodes.h, 17
- src/message.c, 18
- src/message.h, 21
- src/sllist.c, 25
- src/sllist.h, 29
- src/zip-zop-client.c, 34
- src/zip-zop-server.c, 37
- thread
 - client, 5
- zip-zop-client.c
 - check_args, 35

- communicate, [35](#)
- create_and_connect, [36](#)
- get_server_addr, [36](#)
- listen_thread, [36](#)
- MESSAGE_LEN, [35](#)
- main, [36](#)
- PORT, [35](#)
- print_usage, [36](#)
- server_introduction, [36](#)
- show_message, [36](#)
- speak_thread, [37](#)

zip-zop-server.c

- accept_clients, [39](#)
- BACKLOG, [38](#)
- CLIENT_LIST_MUTEX, [42](#)
- CLIENT_LIST, [42](#)
- CLIENT_NAME_LEN, [38](#)
- client_thread_broadcast, [39](#)
- client_thread_listen, [40](#)
- create_and_bind, [40](#)
- create_new_client, [40](#)
- get_internet_addr, [41](#)
- kill_client, [41](#)
- MESSAGE_LEN, [38](#)
- main, [41](#)
- PORT, [39](#)