

qmesg

0.3

Generated by Doxygen 1.15.0

1 Bug List	1
2 Data Structure Index	3
2.1 Data Structures	3
3 File Index	5
3.1 File List	5
4 Data Structure Documentation	7
4.1 _session_t Struct Reference	7
4.2 _user_t Struct Reference	7
4.3 client_message_t Struct Reference	7
5 File Documentation	9
5.1 client_session.h	9
5.2 command.h	9
5.3 common.h	9
5.4 irc.h	10
5.5 user.h	10
5.6 src/linux/main.c File Reference	10
5.6.1 Detailed Description	11
5.7 src/linux/response.c File Reference	12
5.7.1 Detailed Description	12
5.7.2 Function Documentation	13
5.7.2.1 parse()	13
5.7.3 Variable Documentation	13
5.7.3.1 cm_table	13
5.8 src/linux/session.c File Reference	13
5.8.1 Detailed Description	14
5.8.2 Function Documentation	14
5.8.2.1 session_connect()	14
5.8.2.2 session_destroy()	15
5.8.2.3 session_disconnect()	15
5.8.2.4 session_fd()	16
5.8.2.5 session_init()	16
5.9 src/shared/irc.c File Reference	17
5.9.1 Detailed Description	18
5.9.2 Function Documentation	18
5.9.2.1 join()	18
5.9.2.2 list()	19
5.9.2.3 nick()	19
5.9.2.4 part()	20
5.9.2.5 pass()	21
5.9.2.6 ping()	21

5.9.2.7 pong()	22
5.9.2.8 privmsg()	22
5.9.2.9 quit()	22
5.9.2.10 topic()	23
5.9.2.11 user()	23
5.10 src/shared/user.c File Reference	24
5.10.1 Detailed Description	24
5.10.2 Function Documentation	25
5.10.2.1 set_nickname()	25
5.10.2.2 set_password()	25
5.10.2.3 set_realname()	25
5.10.2.4 set_username()	26
5.10.2.5 user_destroy()	26
5.10.2.6 user_init()	26
Index	29

Chapter 1

Bug List

File [irc.c](#)

None known

File [main.c](#)

Possible race conditions during authentication resulting in misses. (maybe sends to fast)

Unexpected cuts in responses from the server

Seems to get stuck in a livelock somewhere resulting in not being able to input commands

File [response.c](#)

None known

File [session.c](#)

None known

File [user.c](#)

None known

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

_session_t	7
_user_t	7
client_message_t	7

Chapter 3

File Index

3.1 File List

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

include/ client_session.h	9
include/ command.h	9
include/ common.h	9
include/ irc.h	10
include/ user.h	10
src/linux/ main.c	
Main	10
src/linux/ response.c	
Provide responses for messages received from a server	12
src/linux/ session.c	
Manage TCP client-side sessions	13
src/shared/ irc.c	
Provide IRC commands for Client to Server communication	17
src/shared/ user.c	
Manage user data	24

Chapter 4

Data Structure Documentation

4.1 `_session_t` Struct Reference

Data Fields

- struct sockaddr_in **adr**
- int **fd**
- int **status**

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

- include/client_session.h

4.2 `_user_t` Struct Reference

Data Fields

- char * **password**
- char * **nickname**
- char * **username**
- char * **realname**

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

- include/user.h

4.3 `client_message_t` Struct Reference

Data Fields

- char * **name**
- int **len**
- int(* **func**)(int fd, char *args)

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

- include/command.h

Chapter 5

File Documentation

5.1 client_session.h

```
00001 #ifndef CLIENT_SESSION_H
00002 #define CLIENT_SESSION_H
00003 // Q: Should these be included here?
00004 // Q: Can I remove some of these?
00005 #include <arpa/inet.h>
00006 #include <netinet/in.h>
00007 #include <fcntl.h>
00008 #include <sys/socket.h>
00009 #include <unistd.h>
00010
00011 #define MAX_ADDR_LEN 100
00012 #define DOMAIN AF_INET // IPv4
00013 #define TYPE SOCK_STREAM // TCP
00014 #define PROTOCOL 0 // use provided protocol configuration
00015
00016 typedef struct _session_t {
00017     struct sockaddr_in adr; // Q: Should this be a pointer?
00018     int fd;
00019     int status; // 0: open connection, 1: closed connection
00020 } session_t;
00021
00022 session_t *session_init(session_t *srv, int prt, char * adr, int adr_len);
00023 extern int session_destroy(session_t *srv);
00024 extern int session_connect(session_t *srv);
00025 extern int session_disconnect(session_t *srv);
00026 extern int session_fd(session_t *sesh);
00027 // TODO: Make save and load function
00028 #endif
00029
```

5.2 command.h

```
00001 #ifndef COMMAND_H
00002 #define COMMAND_H
00003
00004 typedef struct client_message_t {
00005     char * name;
00006     int len;
00007     int (*func)(int fd, char * args);
00008 } client_message;
00009
00010 extern int parse(int fd, char *msg, int len);
00011 #endif
```

5.3 common.h

```
00001 #ifndef COMMON_H
00002 #define COMMON_H
```

```

00003
00004 #define MSG_MAX_LEN 1024
00005 #define PASSWORD_MAX_LEN 30
00006 #define NICKNAME_MAX_LEN 30
00007 #define USERNAME_MAX_LEN 30
00008 #define REALNAME_MAX_LEN 30
00009 #define IP_MAX_LEN 16
00010 #define PORT_MAX_LEN 5
00011 #define CMD_MAX_LEN 8
00012 #endif

```

5.4 irc.h

```

00001 #include <string.h>
00002 #include <unistd.h>
00003 #include <common.h> // Provides common magic literals
00004
00005 extern int pass(int fd, char *password);
00006 extern int nick(int fd, char *nickname);
00007 extern int user(int fd, char *username, char *realname);
00008 extern int ping(int fd, char *token);
00009 extern int pong(int fd, char *token);
00010 extern int join(int fd, char *channel);
00011 extern int quit(int fd);
00012 extern int privmsg(int fd, char *channel, char *msg);
00013 extern int list(int fd);
00014 extern int topic(int fd, char *channel);
00015 extern int part(int fd, char *channel);

```

5.5 user.h

```

00001 #ifndef USER_H
00002 #define USER_H
00003 #define PASSWORD_MAX_LEN 30
00004 #define NICKNAME_MAX_LEN 30
00005 #define USERNAME_MAX_LEN 30
00006 #define REALNAME_MAX_LEN 30
00007
00008 typedef struct _user_t {
00009     char *password;
00010     char *nickname;
00011     char *username;
00012     char *realname;
00013 } user_t;
00014
00015 extern user_t *user_init(user_t *usr, char *psw, char *nck, char *usn, char *rln);
00016 extern void user_destroy(user_t *usr);
00017 // TODO: Make save and load function
00018
00019 // Setters for user values
00020 extern int set_password(user_t *usr, char *psw);
00021 extern int set_nickname(user_t *usr, char *nck);
00022 extern int set_username(user_t *usr, char *usn);
00023 extern int set_realname(user_t *usr, char *rln);
00024
00025 #endif
00026

```

5.6 src/linux/main.c File Reference

Main.

```

#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <signal.h>
#include <stdbool.h>
#include <netinet/in.h>

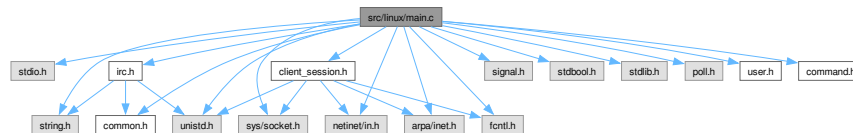
```

```

#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <poll.h>
#include <fcntl.h>
#include <user.h>
#include <irc.h>
#include <client_session.h>
#include <command.h>
#include <common.h>

```

Include dependency graph for main.c:



Macros

- `#define TITLE "qmesg - linux client\n"`
- `#define INTERRUPT_MSG "\nCaught interrupt\n"`
- `#define NUM_FEEDS 2`
- `#define TEST_PASSWORD "psw"`
- `#define TEST_NICKNAME "nck"`
- `#define TEST_USERNAME "usr"`
- `#define TEST_REALNAME "rln"`
- `#define TEST_PORT 6667`
- `#define TEST_ADDR "127.0.0.1"`

Functions

- `int main (int argc, char const *argv[])`

5.6.1 Detailed Description

Main.

Author

Aqiel Oostenbrug

Date

December 3, 2025

Version

1.1

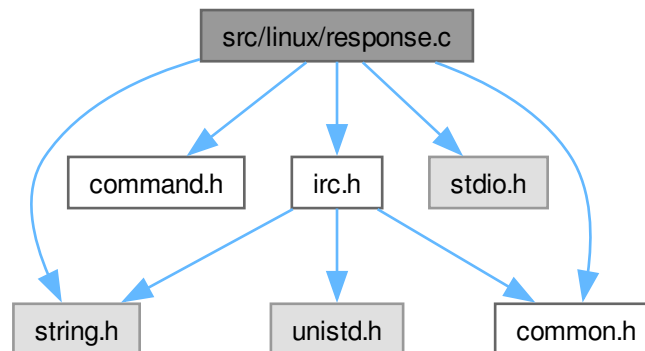
- Bug** Possible race conditions during authentication resulting in misses. (maybe sends to fast)
- Unexpected cuts in responses from the server
- Seems to get stuck in a livelock somewhere resulting in not being able to input commands

5.7 src/linux/response.c File Reference

Provide responses for messages received from a server.

```
#include <string.h>
#include <command.h>
#include <common.h>
#include <stdio.h>
#include <irc.h>
```

Include dependency graph for response.c:



Functions

- int `parse` (int fd, char *response, int response_len)
Execute command corresponding to `response` with `response_len`.

Variables

- client_message `cm_table` []

5.7.1 Detailed Description

Provide responses for messages received from a server.

Author

Aqiel Oostenbrug

Date

December 3, 2025

Version

1.1

Bug None known

5.7.2 Function Documentation

5.7.2.1 parse()

```
int parse (
    int fd,
    char * response,
    int response_len)
```

Execute command corresponding to `response` with `response_len`.

Parameters

<i>fd</i>	file descriptor for the command to write to
<i>response</i>	response to parse
<i>response_len</i>	length of the response to pars

Returns

`cm_table[i].func(fd, params)` if successful
 0 if unmatched,
 -1 if `fd` is invalid,
 -2 if response is invalid,
 -3 if response is too short,
 -4 if response only includes a client message parameter,
 -5 if response only includes a space as a client message parameter

5.7.3 Variable Documentation

5.7.3.1 cm_table

```
client_message cm_table[]
```

Initial value:

```
= {
    {"PING", 4, respond_pong}
}
```

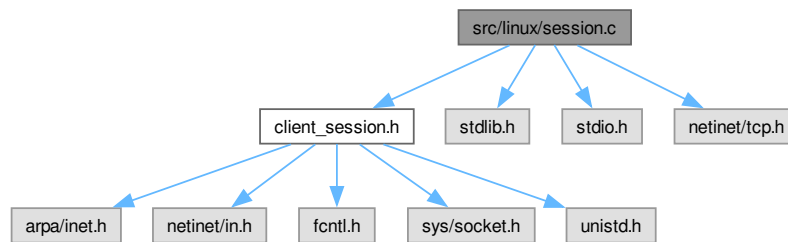
5.8 src/linux/session.c File Reference

Manage TCP client-side sessions.

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

```
#include <netinet/tcp.h>
```

Include dependency graph for session.c:



Functions

- session_t * [session_init](#) (session_t *sesh, int prt, char *adr, int adr_len)
Initialize sesh with prt, adr, and adr_len.
- int [session_destroy](#) (session_t *sesh)
Destroy sesh.
- int [session_connect](#) (session_t *sesh)
Connect to the server specified by sesh.
- int [session_disconnect](#) (session_t *sesh)
Disconnect from the server specified by sesh.
- int [session_fd](#) (session_t *sesh)
Return the file descriptor granted by sesh.

5.8.1 Detailed Description

Manage TCP client-side sessions.

Author

Aqiel Oostenbrug

Date

December 3, 2025

Version

1.0

Bug None known

5.8.2 Function Documentation

5.8.2.1 session_connect()

```
int session_connect (
    session_t * sesh)
```

Connect to the server specified by sesh.

Parameters

<i>sessh</i>	session to start
--------------	------------------

Returns

0 if successful,
1 otherwise

5.8.2.2 session_destroy()

```
int session_destroy (  
    session_t * sessh)
```

Destroy *sessh*.

Parameters

<i>sessh</i>	session to destroy
--------------	--------------------

Returns

0 if successful
1 if *sessh* was not initialized

Here is the call graph for this function:

**5.8.2.3 session_disconnect()**

```
int session_disconnect (  
    session_t * sessh)
```

Disconnect from the server specified by *sessh*.

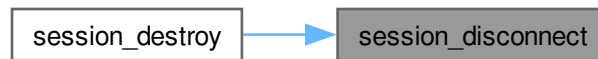
Parameters

<i>sessh</i>	session to end
--------------	----------------

Returns

0 if successful,
1 otherwise

Here is the caller graph for this function:

**5.8.2.4 session_fd()**

```
int session_fd (  
    session_t * sesh)
```

Return the file descriptor granted by *sesh*.

Parameters

<i>sesh</i>	to source the file descriptor from
-------------	------------------------------------

Returns

sesh->fd file descriptor of *sesh*
-1 otherwise

5.8.2.5 session_init()

```
session_t * session_init (  
    session_t * sesh,  
    int prt,  
    char * adr,  
    int adr_len)
```

Initialize *sesh* with *prt*, *adr*, and *adr_len*.

Parameters

<i>sesh</i>	session to initialize
<i>prt</i>	port to connect to
<i>adr</i>	ip address to connect to

<code>adr_len</code>	length of <code>adr</code>
----------------------	----------------------------

Returns

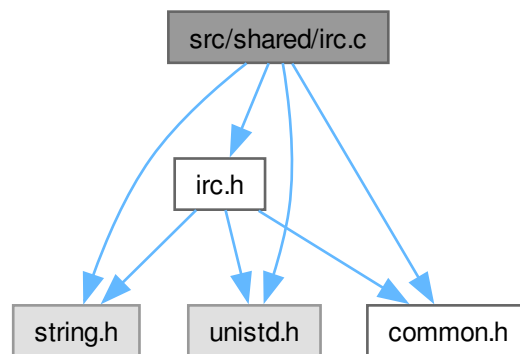
sess if successful,
 NULL otherwise

5.9 src/shared/irc.c File Reference

Provide IRC commands for Client to Server communication.

```
#include <string.h>
#include <unistd.h>
#include <common.h>
#include <irc.h>
```

Include dependency graph for irc.c:

**Functions**

- int `pass` (int fd, char *password)
write password to the server.
- int `nick` (int fd, char *nickname)
write nickname to the server.
- int `user` (int fd, char *username, char *realname)
write username and realname to the server.
- int `ping` (int fd, char *token)
Ping the server.
- int `pong` (int fd, char *token)
Pong the server.
- int `join` (int fd, char *channel)
Join server.

- int `quit` (int fd)
Terminate the connection to the server.
- int `privmsg` (int fd, char *channel, char *msg)
write msg channel to the server.
- int `list` (int fd)
Request the list of channels.
- int `topic` (int fd, char *channel)
Request the topic of channel.
- int `part` (int fd, char *channel)
Request to leave channel.

5.9.1 Detailed Description

Provide IRC commands for Client to Server communication.

Note that since write is used no fflush is required!

Author

Aqiel Oostenbrug

Date

November 26, 2025

Version

1.1

Bug None known

See also

<https://modern.ircdocs.horse/>

5.9.2 Function Documentation

5.9.2.1 join()

```
int join (
    int fd,
    char * channel)
```

Join server.

Parameters

in	<i>fd</i>	file descriptor to write to
in	<i>channel</i>	channel to join

Returns

0 if successful,
1 if *fd* is invalid,
3 if *channel* is invalid,
4 if message is too long

5.9.2.2 list()

```
int list (  
    int fd)
```

Request the list of channels.

Parameters

in	<i>fd</i>	file descriptor to write to
----	-----------	-----------------------------

Returns

0 if request send successfully,
1 if *fd* is invalid

5.9.2.3 nick()

```
int nick (  
    int fd,  
    char * nickname)
```

write *nickname* to the server.

Parameters

in	<i>fd</i>	file descriptor to write to
in	<i>nickname</i>	nickname to write

Returns

0 if successful,
1 if *fd* is invalid,
2 if *nickname* is invalid,
3 if message is too long

5.9.2.4 `part()`

```
int part (  
    int fd,  
    char * channel)
```

Request to leave `channel`.

Parameters

in	<i>fd</i>	file descriptor to write to
in	<i>channel</i>	channel to leave

Returns

0 if successful,
1 if *fd* is invalid,
2 if *channel* is invalid,
3 if total message is too long

5.9.2.5 pass()

```
int pass (  
    int fd,  
    char * password)
```

write *password* to the server.

Parameters

in	<i>fd</i>	file descriptor to write to
in	<i>password</i>	password to write

Returns

0 if successful,
1 if *fd* is invalid,
2 if *password* is invalid,
3 if message is too long

5.9.2.6 ping()

```
int ping (  
    int fd,  
    char * token)
```

Ping the server.

Parameters

in	<i>fd</i>	file descriptor to write to
in	<i>token</i>	token to write

Returns

0 if successful,
1 if *fd* is invalid,
3 if *token* is invalid,
4 if message is too long

5.9.2.7 pong()

```
int pong (
    int fd,
    char * token)
```

Pong the server.

Parameters

in	<i>fd</i>	file descriptor to write to
in	<i>token</i>	token to write

Returns

0 if successful,
1 if *fd* is invalid,
3 if *token* is invalid,
4 if message is too long

5.9.2.8 privmsg()

```
int privmsg (
    int fd,
    char * channel,
    char * msg)
```

write *msg* *channel* to the server.

Parameters

	<i>fd</i>	file descriptor to write to
in	<i>channel</i>	channel to write
in	<i>msg</i>	message to write

Returns

0 if request send successfully,
1 if *fd* is invalid,
2 if *channel* is invalid,
3 if *msg* is invalid,
4 if total message is too long

5.9.2.9 quit()

```
int quit (
    int fd)
```

Terminate the connection to the server.

Parameters

in	<i>fd</i>	file descriptor to write to
in	<i>token</i>	token to write

Returns

0 if successful,
1 if *fd* is invalid

5.9.2.10 topic()

```
int topic (  
    int fd,  
    char * channel)
```

Request the topic of *channel*.

Parameters

in	<i>fd</i>	file descriptor to write to
in	<i>channel</i>	channel from which the topic to request

Returns

0 if successful,
1 if *fd* is invalid,
2 if *channel* is invalid,
3 if total message is too long

5.9.2.11 user()

```
int user (  
    int fd,  
    char * username,  
    char * realname)
```

write *username* and *realname* to the server.

Parameters

	<i>fd</i>	file descriptor to write to
in	<i>username</i>	username to write
in	<i>realname</i>	realname to write

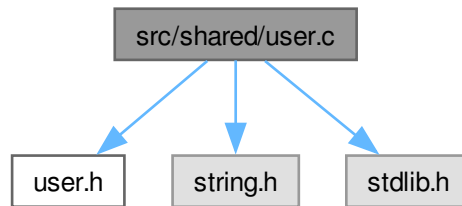
Returns

0 if successful,
1 if *fd* is invalid,
2 if *username* is invalid,
3 if *realname* is invalid,
4 if message is too long

5.10 src/shared/user.c File Reference

Manage user data.

```
#include <user.h>
#include <string.h>
#include <stdlib.h>
Include dependency graph for user.c:
```



Functions

- `user_t * user_init (user_t *usr, char *psw, char *nck, char *usn, char *rln)`
Initialize `usr` with `psw`, `pwt`, `adr`, and `adr_len`.
- `void user_destroy (user_t *usr)`
Destroy `usr`.
- `int set_password (user_t *usr, char *psw)`
Set `psw` to `usr`.
- `int set_nickname (user_t *usr, char *nck)`
Set `nck` to `usr`.
- `int set_username (user_t *usr, char *usn)`
Set `usn` to `usr`.
- `int set_realname (user_t *usr, char *rln)`
Set `rln` to `usr`.

5.10.1 Detailed Description

Manage user data.

Author

Aqiel Oostenbrug

Date

December 3, 2025

Version

1.0

Bug None known

5.10.2 Function Documentation

5.10.2.1 set_nickname()

```
int set_nickname (  
    user_t * usr,  
    char * nck)
```

Set *nck* to *usr*.

Parameters

<i>usr</i>	user to update
<i>nck</i>	nickname to set

Returns

0 if successful, 1 otherwise

5.10.2.2 set_password()

```
int set_password (  
    user_t * usr,  
    char * psw)
```

Set *psw* to *usr*.

Parameters

<i>usr</i>	user to update
<i>psw</i>	password to set

Returns

0 if successful, 1 otherwise

5.10.2.3 set_realname()

```
int set_realname (  
    user_t * usr,  
    char * rln)
```

Set *rln* to *usr*.

Parameters

<i>usr</i>	user to update
<i>rln</i>	realname to set

Returns

0 if successful, 1 otherwise

5.10.2.4 set_username()

```
int set_username (  
    user_t * usr,  
    char * usn)
```

Set *usn* to *usr*.

Parameters

<i>usr</i>	user to update
<i>usn</i>	username to set

Returns

0 if successful, 1 otherwise

5.10.2.5 user_destroy()

```
void user_destroy (  
    user_t * usr)
```

Destroy *usr*.

Parameters

<i>usr</i>	user to destroy
------------	-----------------

5.10.2.6 user_init()

```
user_t * user_init (  
    user_t * usr,  
    char * psw,  
    char * nck,  
    char * usn,  
    char * rln)
```

Initialize *usr* with *psw*, *prt*, *adr*, and *adr_len*.

Parameters

<i>usr</i>	user to initialize
<i>psw</i>	password of user
<i>nck</i>	nickname of user
<i>usn</i>	username of user
<i>rln</i>	realname of user

Returns

`usr` if successful, `NULL` otherwise

Index

[_session_t](#), 7

[_user_t](#), 7

[Bug List](#), 1

[client_message_t](#), 7

[cm_table](#)
 [response.c](#), 13

[include/client_session.h](#), 9

[include/command.h](#), 9

[include/common.h](#), 9

[include/irc.h](#), 10

[include/user.h](#), 10

[irc.c](#)
 [join](#), 18
 [list](#), 19
 [nick](#), 19
 [part](#), 19
 [pass](#), 21
 [ping](#), 21
 [pong](#), 21
 [privmsg](#), 22
 [quit](#), 22
 [topic](#), 23
 [user](#), 23

[join](#)
 [irc.c](#), 18

[list](#)
 [irc.c](#), 19

[nick](#)
 [irc.c](#), 19

[parse](#)
 [response.c](#), 13

[part](#)
 [irc.c](#), 19

[pass](#)
 [irc.c](#), 21

[ping](#)
 [irc.c](#), 21

[pong](#)
 [irc.c](#), 21

[privmsg](#)
 [irc.c](#), 22

[quit](#)
 [irc.c](#), 22

[response.c](#)
 [cm_table](#), 13
 [parse](#), 13

[session.c](#)
 [session_connect](#), 14
 [session_destroy](#), 15
 [session_disconnect](#), 15
 [session_fd](#), 16
 [session_init](#), 16

[session_connect](#)
 [session.c](#), 14

[session_destroy](#)
 [session.c](#), 15

[session_disconnect](#)
 [session.c](#), 15

[session_fd](#)
 [session.c](#), 16

[session_init](#)
 [session.c](#), 16

[set_nickname](#)
 [user.c](#), 25

[set_password](#)
 [user.c](#), 25

[set_realname](#)
 [user.c](#), 25

[set_username](#)
 [user.c](#), 26

[src/linux/main.c](#), 10

[src/linux/response.c](#), 12

[src/linux/session.c](#), 13

[src/shared/irc.c](#), 17

[src/shared/user.c](#), 24

[topic](#)
 [irc.c](#), 23

[user](#)
 [irc.c](#), 23

[user.c](#)
 [set_nickname](#), 25
 [set_password](#), 25
 [set_realname](#), 25
 [set_username](#), 26
 [user_destroy](#), 26
 [user_init](#), 26

[user_destroy](#)
 [user.c](#), 26

[user_init](#)
 [user.c](#), 26