Elevator Project

Generated by Doxygen 1.8.13

Contents

Chapter 1

Elevator Project - TTK4235 Embedded Systems

The main goal of this project was to design and program a functional elevator that can receive hall orders up, hall orders down, and cab call from within the elevator. The project was programmed in C utilizing the elevator hardware found in the Real time programming laboratory. The project can also be run on the `SimElevatorServer` program to test the program.

With permission from the Lab Instructor Kolbjørn Austreng, we were permitted to communicate with the elevator hardware via the a server used in the course TTK4145 Real-time Programming, `ElevatorServer`. This means `elevator_hardware.c` and `elevator_hardware.h` were used to communicate with the elevator instead of:

- `elev.c`
- `elev.h`
- `io.c`
- `io.h`

The functions names in `elevator_hardware.c` are the same and behave in the same manner as the files listed above.

Documentation

Documentation for this project can be found as html version to be opened in an internet browser or via the pdf document. The html documentation can be found by opening `elevator/html/index.html` (or by clicking `here`) and the `pdf` version can be found by opening `elevator/latex/refman.pdf`(or by clicking `here`). The various diagrams for the project can be found in the `elevator/docs/` folder and can also be seen below. It is **HIGHLY RECOMMENDED** to view the `html` documentation instead of the `pdf` version as the formatting makes it easier to read.

Running the program

The program can be run in the lab by starting up a terminal by typing in the command:

```
ElevatorServer
```

In another terminal instance the following can be written to compile and run the elevator:

```
make
./heis
```

Alternatively, the following command can be run once to grant permission to a bash script:

```
chmod +x run_elevator
```

Followed by the following command every time the program is to be compiled and run:

```
./run_elevator
```

Diagrams

The following diagrams were created to more easily design and understand the system. These can also be found as `.pdf` versions in the `elevator/docs/` folder

Chapter 2

File Index

2.1 File List

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

Chapter 3

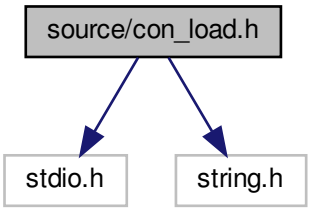
File Documentation

3.1 source/con_load.h File Reference

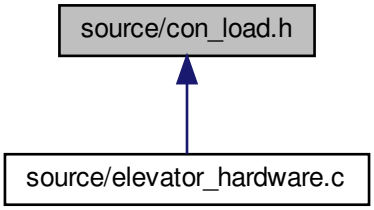
Background file used by elevator_hardware.h to communicate with the elevator.

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

Include dependency graph for con_load.h:



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



3.1.1 Detailed Description

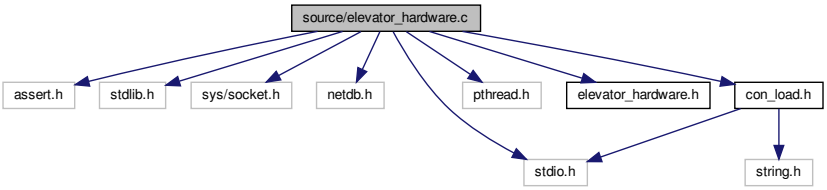
Background file used by elevator_hardware.h to communicate with the elevator.

3.2 source/elevator_hardware.c File Reference

Implementation of the functions in elevator_hardware.h.

```
#include <assert.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netdb.h>
#include <stdio.h>
#include <pthread.h>
#include "elevator_hardware.h"
#include "con_load.h"
```

Include dependency graph for elevator_hardware.c:



Functions

- void elev_init ()
- void elev_set_motor_direction (elev_motor_direction_t dirn)
- void elev_set_button_lamp (elev_button_type_t button, int floor, int value)
- void elev_set_floor_indicator (int floor)
- void elev_set_door_open_lamp (int value)
- void elev_set_stop_lamp (int value)
- int elev_get_button_signal (elev_button_type_t button, int floor)
- int elev_get_floor_sensor_signal (void)
- int elev_get_stop_signal (void)
- int elev_get_obstruction_signal (void)

3.2.1 Detailed Description

Implementation of the functions in elevator_hardware.h.

3.2.2 Function Documentation

3.2.2.1 elev_get_button_signal()

```
int elev_get_button_signal (
            elev_button_type_t button,
            int floor )
```

Gets a button signal.

Parameters

in	*button*	Which button type to check as defined in elev_button_type_t.
in	*floor*	Which floor to check button. Must be 0-3.

Returns

0 if button is not pushed. 1 if button is pushed.

Definition at line 96 of file elevator_hardware.c.

3.2.2.2 elev_get_floor_sensor_signal()

```
int elev_get_floor_sensor_signal (
            void  )
```

Get floor sensor signal.

Returns

-1 if elevator is not on a floor. 0-3 if elevator is on floor. 0 is ground floor, 3 is top floor.

Definition at line 106 of file elevator_hardware.c.

3.2.2.3 elev_get_obstruction_signal()

```
int elev_get_obstruction_signal (
            void  )
```

Get signal from obstruction switch.

Returns

1 if obstruction is enabled. 0 if not.

Definition at line 126 of file elevator_hardware.c.

3.2.2.4 elev_get_stop_signal()

```
int elev_get_stop_signal (
            void  )
```

Get signal from stop button.

Returns

1 if stop button is pushed, 0 if not.

Definition at line 116 of file elevator_hardware.c.

3.2.2.5 elev_init()

```
void elev_init ( )
```

Initialize elevator.

Returns

Non-zero on success, 0 on failure.

Definition at line 19 of file elevator_hardware.c.

3.2.2.6 elev_set_button_lamp()

```
void elev_set_button_lamp (
            elev_button_type_t button,
            int floor,
            int value )
```

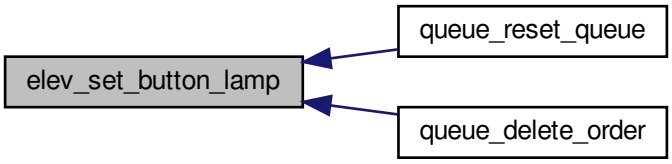
Set a button lamp.

Parameters

in	*lamp*	Which type of lamp to set as defined in elev_button_type_t.
in	*floor*	Floor of lamp to set. Must be 0-3
in	*value*	Non-zero value turns lamp on, 0 turns lamp off.

Definition at line 58 of file elevator_hardware.c.

Here is the caller graph for this function:



3.2.2.7 elev_set_door_open_lamp()

```
void elev_set_door_open_lamp (
            int value )
```

Turn door-open lamp on or off.

Parameters

in	*value*	Non-zero value turns lamp on, 0 turns lamp off.

Definition at line 80 of file elevator_hardware.c.

3.2.2.8 elev_set_floor_indicator()

```
void elev_set_floor_indicator (
            int floor )
```

Set floor indicator lamp for a given floor.

Parameters

in	*floor*	Which floor lamp to turn on. Other floor lamps are turned off.

Definition at line 70 of file elevator_hardware.c.

3.2.2.9 elev_set_motor_direction()

```
void elev_set_motor_direction (
            elev_motor_direction_t dirn )
```

Sets the motor direction of the elevator.

Parameters

in	*dirn*	New direction of the elevator.

Definition at line 51 of file elevator_hardware.c.

3.2.2.10 elev_set_stop_lamp()

```
void elev_set_stop_lamp (
            int value )
```

Turn stop lamp on or off.

Parameters

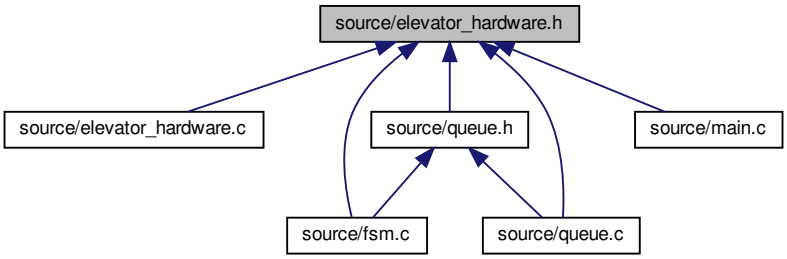
in	*value*	Non-zero value turns lamp on, 0 turns lamp off.

Definition at line 87 of file elevator_hardware.c.

3.3 source/elevator_hardware.h File Reference

The driver that communicates with the elevator hardware. This is done through a server that is set up on a port on the machine and communicates with the elevator hardware. The function interactions are identical to the driver provided by the lab instructor in the Embedded Systems course.

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



Macros

- #define N_FLOORS 4

  Number of floors in floor_t without ORDER_FLOOR_UNKNOWN, also Hardware-dependent.
- #define N_BUTTONS 3

  Number of button types in elev_button_type_t.

Enumerations

- enum elev_motor_direction_t { DIRN_DOWN = -1, DIRN_STOP = 0, DIRN_UP = 1 }
- enum elev_button_type_t { BUTTON_CALL_UP = 0, BUTTON_CALL_DOWN = 1, BUTTON_COMMAND = 2 }

Functions

- void elev_init ()
- void elev_set_motor_direction (elev_motor_direction_t dirn)
- void elev_set_button_lamp (elev_button_type_t button, int floor, int value)
- void elev_set_floor_indicator (int floor)
- void elev_set_door_open_lamp (int value)
- void elev_set_stop_lamp (int value)
- int elev_get_button_signal (elev_button_type_t button, int floor)
- int elev_get_floor_sensor_signal (void)
- int elev_get_stop_signal (void)
- int elev_get_obstruction_signal (void)

3.3.1 Detailed Description

The driver that communicates with the elevator hardware. This is done through a server that is set up on a port on the machine and communicates with the elevator hardware. The function interactions are identical to the driver provided by the lab instructor in the Embedded Systems course.

3.3.2 Enumeration Type Documentation

3.3.2.1 elev_button_type_t

`enum elev_button_type_t`

Button types for function elev_set_button_lamp() and elev_get_button().

Enumerator

BUTTON_CALL_UP	Elevator hall order in upwards direction.
BUTTON_CALL_DOWN	Elevator hall order in downwards direction.
BUTTON_COMMAND	Elevator cab order from within the elevator.

Definition at line 30 of file elevator_hardware.h.

3.3.2.2 elev_motor_direction_t

`enum elev_motor_direction_t`

Motor direction for function elev_set_motor_direction().

Enumerator

DIRN_DOWN	Elevator motor direction downwards.
DIRN_STOP	Elevator motor stopped.
DIRN_UP	Elevator motor direction upwards.

Definition at line 21 of file elevator_hardware.h.

3.3.3 Function Documentation

3.3.3.1 elev_get_button_signal()

```
int elev_get_button_signal (
            elev_button_type_t button,
            int floor )
```

Gets a button signal.

Parameters

in	*button*	Which button type to check as defined in elev_button_type_t.
in	*floor*	Which floor to check button. Must be 0-3.

Returns

0 if button is not pushed. 1 if button is pushed.

Definition at line 96 of file elevator_hardware.c.

3.3.3.2 elev_get_floor_sensor_signal()

```
int elev_get_floor_sensor_signal (
            void  )
```

Get floor sensor signal.

Returns

-1 if elevator is not on a floor. 0-3 if elevator is on floor. 0 is ground floor, 3 is top floor.

Definition at line 106 of file elevator_hardware.c.

3.3.3.3 elev_get_obstruction_signal()

```
int elev_get_obstruction_signal (
            void  )
```

Get signal from obstruction switch.

Returns

1 if obstruction is enabled. 0 if not.

Definition at line 126 of file elevator_hardware.c.

3.3.3.4 elev_get_stop_signal()

```
int elev_get_stop_signal (
            void  )
```

Get signal from stop button.

Returns

1 if stop button is pushed, 0 if not.

Definition at line 116 of file elevator_hardware.c.

3.3.3.5 elev_init()

```
void elev_init ( )
```

Initialize elevator.

Returns

Non-zero on success, 0 on failure.

Definition at line 19 of file elevator_hardware.c.

3.3.3.6 elev_set_button_lamp()

```
void elev_set_button_lamp (
            elev_button_type_t button,
            int floor,
            int value )
```

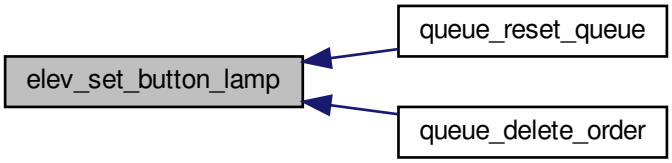
Set a button lamp.

Parameters

in	*lamp*	Which type of lamp to set as defined in elev_button_type_t.
in	*floor*	Floor of lamp to set. Must be 0-3
in	*value*	Non-zero value turns lamp on, 0 turns lamp off.

Definition at line 58 of file elevator_hardware.c.

Here is the caller graph for this function:



3.3.3.7 elev_set_door_open_lamp()

```
void elev_set_door_open_lamp (
            int value )
```

Turn door-open lamp on or off.

Parameters

in	*value*	Non-zero value turns lamp on, 0 turns lamp off.

Definition at line 80 of file elevator_hardware.c.

3.3.3.8 elev_set_floor_indicator()

```
void elev_set_floor_indicator (
            int floor )
```

Set floor indicator lamp for a given floor.

Parameters

in	*floor*	Which floor lamp to turn on. Other floor lamps are turned off.

Definition at line 70 of file elevator_hardware.c.

3.3.3.9 elev_set_motor_direction()

```
void elev_set_motor_direction (
            elev_motor_direction_t dirn )
```

Sets the motor direction of the elevator.

Parameters

in	*dirn*	New direction of the elevator.

Definition at line 51 of file elevator_hardware.c.

3.3.3.10 elev_set_stop_lamp()

```
void elev_set_stop_lamp (
            int value )
```

Turn stop lamp on or off.

Parameters

in	*value*	Non-zero value turns lamp on, 0 turns lamp off.

Definition at line 87 of file elevator_hardware.c.

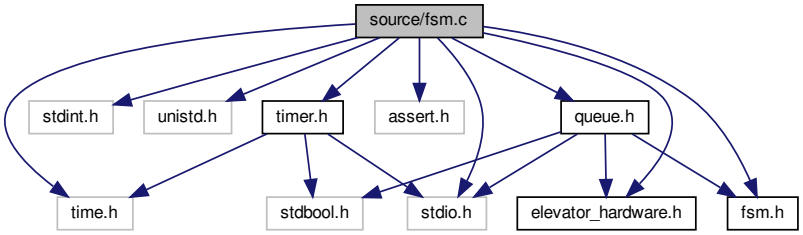
3.4 source/fsm.c File Reference

Implementation of the functions in fsm.h.

```
#include <stdio.h>
#include <stdint.h>
#include <unistd.h>
#include <time.h>
#include <assert.h>
#include "elevator_hardware.h"
#include "fsm.h"
#include "queue.h"
```

```
#include "timer.h"
```

Include dependency graph for fsm.c:



Functions

- void fsm ()

  The functions main responsibilites is managing the finite state machine. Before entering a switch that manages the state machine it polls:

Variables

- static position_t fsm_position = UNKNOWN

  Local fsm variable used to keep track of the elevators position.
- static floor_t fsm_floor = ORDER_FLOOR_UNKNOWN

  Local fsm variable used to keep track of the elevators last or current floor.
- static time_t fsm_timestamp = 0

  Local fsm variable to keep track of timer responisble for opening the door.
- static elev_motor_direction_t fsm_direction = DIRN_UP

  Local fsm variable used to keep track of the elevators direction of travel. This can never be DIRN_STOP.
- static state_t fsm_state = INIT

  Local fsm variable used to keep track of the elevators state.

3.4.1 Detailed Description

Implementation of the functions in fsm.h.

3.4.2 Function Documentation

3.4.2.1 fsm()

```
void fsm ( )
```

The functions main responsibilites is managing the finite state machine. Before entering a switch that manages the state machine it polls:

- The order buttons
- floor sensors
- stop button

The state machine manages the states in the state_t. The behaviour between the states is described in the state diagram and the sequence diagram.

Parameters

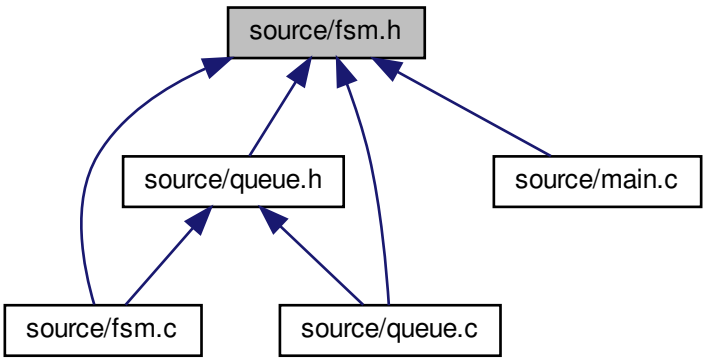
out	*fsm_position*	Elevator position
out	*fsm_floor*	Elevator floor
out	*fsm_timestamp*	Timestamp
out	*fsm_direction*	Elevator direction of travel. This will only ever by DIRN_UP or DIRN_DOWN of elev_motor_direction_t.
out	*fsm_state*	Elevator state

Definition at line 37 of file fsm.c.

3.5 source/fsm.h File Reference

This class that controls the main functions of the elevator. It is responsible for checking input signals in addtion to managing the state of the elevator.

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



Macros

- #define N_POSITIONS 7

  Number of positions in position_t without UNKNOWN.

Enumerations

- enum position_t {
  FLOOR_0, BETWEEN_0_AND_1, FLOOR_1, BETWEEN_1_AND_2,
  FLOOR_2, BETWEEN_2_AND_3, FLOOR_3, UNKNOWN }
- enum floor_t {
  ORDER_FLOOR_0, ORDER_FLOOR_1, ORDER_FLOOR_2, ORDER_FLOOR_3,
  ORDER_FLOOR_UNKNOWN }
- enum state_t {
  INIT, IDLE, MOVING, OPEN_DOOR,
  EMERGENCY_STOP }

Functions

- void fsm ()

  The functions main responsibilites is managing the finite state machine. Before entering a switch that manages the state machine it polls:

3.5.1 Detailed Description

This class that controls the main functions of the elevator. It is responsible for checking input signals in addtion to managing the state of the elevator.

3.5.2 Enumeration Type Documentation

3.5.2.1 floor_t

```
enum floor_t
```

Floor type to dissern the different floors of the elevator. Mainly used when refering queue_array in queue.h

Enumerator

ORDER_FLOOR_0	Floor 0.
ORDER_FLOOR_1	Floor 1.
ORDER_FLOOR_2	Floor 2.
ORDER_FLOOR_3	Floor 3.
ORDER_FLOOR_UNKNOWN	Unknown floor only during initialization.

Definition at line 30 of file fsm.h.

3.5.2.2 position_t

```
enum position_t
```

Position type to keep track of the position of the elevator.

Enumerator

FLOOR_0	Elevator at floor 0.
BETWEEN_0_AND_1	Elevator between floor 0 and floor 1.
FLOOR_1	Elevator at floor 1.
BETWEEN_1_AND_2	Elevator between floor 1 and floor 2.
FLOOR_2	Elevator at floor 2.
BETWEEN_2_AND_3	Elevator between floor 2 and floor 3.
FLOOR_3	Elevator at floor 3.
UNKNOWN	Unknown position only during initialization.

Definition at line 16 of file fsm.h.

3.5.2.3 state_t

```
enum state_t
```

State types for function fsm().

Enumerator

INIT	Initialization state, only during start up.
IDLE	Idle state, where the elevator is not moving and checks for new orders.
MOVING	Moving either up or down.
OPEN_DOOR	Open door state where the elevator is not moving and opens the door.
EMERGENCY_STOP	Emergency stop state, regardless of position.

Definition at line 41 of file fsm.h.

3.5.3 Function Documentation

3.5.3.1 fsm()

```
void fsm ( )
```

The functions main responsibilites is managing the finite state machine. Before entering a switch that manages the state machine it polls:

- The order buttons
- floor sensors
- stop button

The state machine manages the states in the state_t. The behaviour between the states is described in the state diagram and the sequence diagram.

Parameters

out	*fsm_position*	Elevator position
out	*fsm_floor*	Elevator floor
out	*fsm_timestamp*	Timestamp
out	*fsm_direction*	Elevator direction of travel. This will only ever by DIRN_UP or DIRN_DOWN of elev_motor_direction_t.
out	*fsm_state*	Elevator state

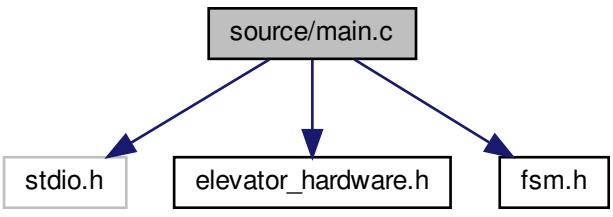
Definition at line 37 of file fsm.c.

3.6 source/main.c File Reference

The main file of the application.

```
#include <stdio.h>
#include "elevator_hardware.h"
#include "fsm.h"
```

Include dependency graph for main.c:



3.6.1 Detailed Description

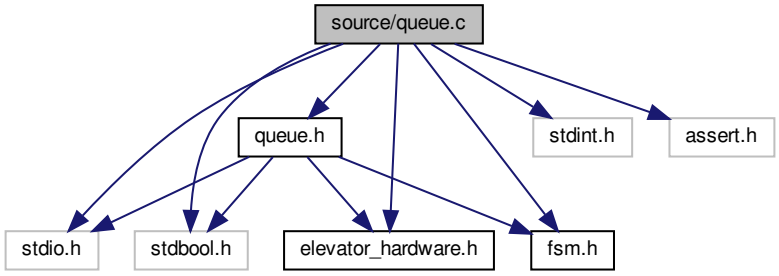
The main file of the application.

3.7 source/queue.c File Reference

Implementation of the functions in queue.h.

```
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <assert.h>
#include "queue.h"
#include "elevator_hardware.h"
#include "fsm.h"
```

Include dependency graph for queue.c:



Functions

- void queue_reset_queue ()

  Deletes all orders in the queue by setting all the order options to the initial value 0.
- void queue_delete_order (floor_t floor)

  Deletes an order in the queue.
- void queue_set_order (elev_button_type_t button, floor_t floor)

  Sets an order in the queue.
- bool queue_is_queue_empty ()

  Checks whether the queue has any orders.
- elev_motor_direction_t queue_get_next_direction (position_t current_position, elev_motor_direction_t last↩ _direction)

  Gets the next direction of the elevator based on the requested priorities.
- bool queue_should_stop (position_t fsm_position, floor_t fsm_floor, elev_motor_direction_t fsm_direction)

  Checks if the elevator should stop when arriving at a floor, based on orders in the queue array. This function will inform the elevator that it should stop if:

Variables

- static int queue_array [N_BUTTONS][N_FLOORS]

  A two dimensional array to keep track of the elevators orders.

3.7.1 Detailed Description

Implementation of the functions in queue.h.

3.7.2 Function Documentation

3.7.2.1 queue_delete_order()

```
void queue_delete_order (
            floor_t floor )
```

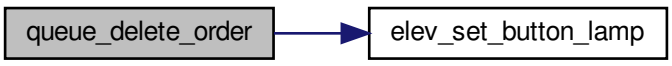
Deletes an order in the queue.

Parameters

in	*floor*	Floor the elevator is at
out	*queue_array*	Queue table

Definition at line 35 of file queue.c.

Here is the call graph for this function:



3.7.2.2 queue_get_next_direction()

```
elev_motor_direction_t queue_get_next_direction (
            position_t current_position,
            elev_motor_direction_t last_direction )
```

Gets the next direction of the elevator based on the requested priorities.

Parameters

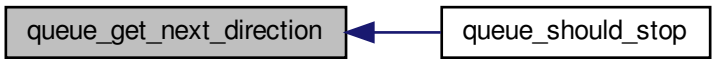
in	*current_position*	At a floor or between floors
in	*last_direction*	The last direction of the elevator

Returns

next direction of the elevator.

Definition at line 64 of file queue.c.

Here is the caller graph for this function:



3.7.2.3 queue_is_queue_empty()

```
bool queue_is_queue_empty ( )
```

Checks whether the queue has any orders.

Returns

1 if the queue is empty, 0 if not.

Definition at line 48 of file queue.c.

3.7.2.4 queue_reset_queue()

```
void queue_reset_queue ( )
```

Deletes all orders in the queue by setting all the order options to the initial value 0.

Parameters

out	*queue_array*	Queue table

Definition at line 24 of file queue.c.

Here is the call graph for this function:

3.7.2.5 queue_set_order()

```
void queue_set_order (
            elev_button_type_t button,
            floor_t floor )
```

Sets an order in the queue.

Parameters

in	*button*	Hardware buttons
in	*floor*	At a floor
out	*queue_array*	Queue table

Definition at line 43 of file queue.c.

3.7.2.6 queue_should_stop()

```
bool queue_should_stop (
            position_t fsm_position,
            floor_t fsm_floor,
            elev_motor_direction_t fsm_direction )
```

Checks if the elevator should stop when arriving at a floor, based on orders in the queue array. This function will inform the elevator that it should stop if:

- There are no further orders in the direction.
- There are cab or hall calls in the direction of travel.

Parameters

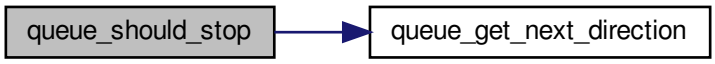
in	*fsm_position*	The current position of the elevator.
in	*fsm_floor*	The current floor of the elevator.
in	*fsm_direction*	The direction of travel the elevator.

Returns

1 of the elevator should stop, 0 if not.

Definition at line 88 of file queue.c.

Here is the call graph for this function:

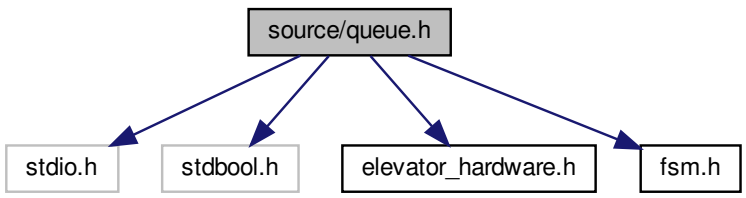


3.8 source/queue.h File Reference

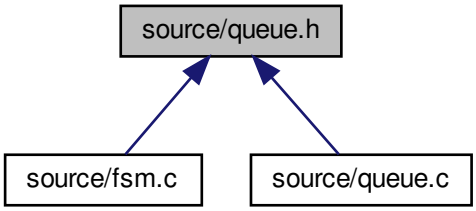
A queue system that helps the finite state machine (fsm) to carry out the orders received from the elevator hardware.

```
#include <stdio.h>
#include <stdbool.h>
#include "elevator_hardware.h"
#include "fsm.h"
```

Include dependency graph for queue.h:



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



Functions

- void queue_reset_queue ()

  Deletes all orders in the queue by setting all the order options to the initial value 0.
- void queue_delete_order (floor_t floor)

  Deletes an order in the queue.
- void queue_set_order (elev_button_type_t button, floor_t floor)

  Sets an order in the queue.
- bool queue_is_queue_empty ()

  Checks whether the queue has any orders.
- elev_motor_direction_t queue_get_next_direction (position_t current_position, elev_motor_direction_t last↩ _direction)

  Gets the next direction of the elevator based on the requested priorities.
- bool queue_should_stop (position_t fsm_position, floor_t fsm_floor, elev_motor_direction_t fsm_direction)

  Checks if the elevator should stop when arriving at a floor, based on orders in the queue array. This function will inform the elevator that it should stop if:

3.8.1 Detailed Description

A queue system that helps the finite state machine (fsm) to carry out the orders received from the elevator hardware.

3.8.2 Function Documentation

3.8.2.1 queue_delete_order()

```
void queue_delete_order (
            floor_t floor )
```

Deletes an order in the queue.

Parameters

in	*floor*	Floor the elevator is at
out	*queue_array*	Queue table

Definition at line 35 of file queue.c.

Here is the call graph for this function:

3.8.2.2 queue_get_next_direction()

```
elev_motor_direction_t queue_get_next_direction (
            position_t current_position,
            elev_motor_direction_t last_direction )
```

Gets the next direction of the elevator based on the requested priorities.

Parameters

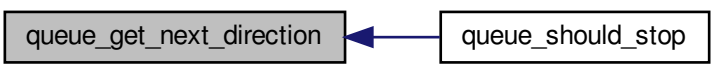
in	*current_position*	At a floor or between floors
in	*last_direction*	The last direction of the elevator

Returns

next direction of the elevator.

Definition at line 64 of file queue.c.

Here is the caller graph for this function:



3.8.2.3 queue_is_queue_empty()

```
bool queue_is_queue_empty ( )
```

Checks whether the queue has any orders.

Returns

1 if the queue is empty, 0 if not.

Definition at line 48 of file queue.c.

3.8.2.4 queue_reset_queue()

```
void queue_reset_queue ( )
```

Deletes all orders in the queue by setting all the order options to the initial value 0.

Parameters

out	*queue_array*	Queue table

Definition at line 24 of file queue.c.

Here is the call graph for this function:

3.8.2.5 queue_set_order()

```
void queue_set_order (
            elev_button_type_t button,
            floor_t floor )
```

Sets an order in the queue.

Parameters

in	*button*	Hardware buttons
in	*floor*	At a floor
out	*queue_array*	Queue table

Definition at line 43 of file queue.c.

3.8.2.6 queue_should_stop()

```
bool queue_should_stop (
            position_t fsm_position,
            floor_t fsm_floor,
            elev_motor_direction_t fsm_direction )
```

Checks if the elevator should stop when arriving at a floor, based on orders in the queue array. This function will inform the elevator that it should stop if:

- There are no further orders in the direction.
- There are cab or hall calls in the direction of travel.

Parameters

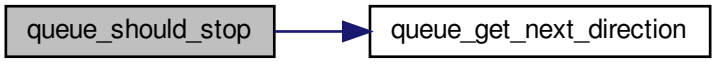
in	*fsm_position*	The current position of the elevator.
in	*fsm_floor*	The current floor of the elevator.
in	*fsm_direction*	The direction of travel the elevator.

Returns

1 of the elevator should stop, 0 if not.

Definition at line 88 of file queue.c.

Here is the call graph for this function:

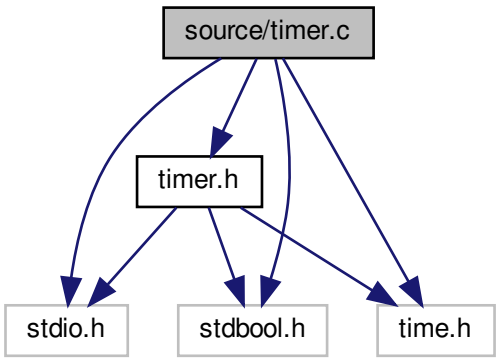


3.9 source/timer.c File Reference

Implementation of the functions in timer.h.

```
#include <stdio.h>
#include <stdbool.h>
#include <time.h>
#include "timer.h"
```

Include dependency graph for timer.c:



Functions

- time_t timer_start_timer ()

  Starts a fictious timer by returning timestamp of the current time of the system.
- bool timer_is_timer_expired (time_t start_timestamp)

  Check whether 3 seconds have passed since the timer started.

3.9.1 Detailed Description

Implementation of the functions in timer.h.

3.9.2 Function Documentation

3.9.2.1 timer_is_timer_expired()

```
bool timer_is_timer_expired (
            time_t start_timestamp )
```

Check whether 3 seconds have passed since the timer started.

Parameters

in	*start_time*	Start time of timer.

Returns

1 if the timer is expired, 0 if not.

Definition at line 17 of file timer.c.

3.9.2.2 timer_start_timer()

```
time_t timer_start_timer ( )
```

Starts a fictious timer by returning timestamp of the current time of the system.

Returns

the start time.

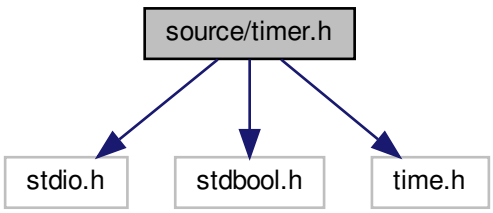
Definition at line 12 of file timer.c.

3.10 source/timer.h File Reference

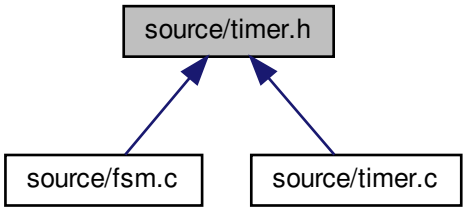
A smaller module that manages the time dependent operations of the state machine.

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

Include dependency graph for timer.h:



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



Functions

- time_t timer_start_timer ()

  Starts a fictious timer by returning timestamp of the current time of the system.
- bool timer_is_timer_expired (time_t start_timestamp)

  Check whether 3 seconds have passed since the timer started.

3.10.1 Detailed Description

A smaller module that manages the time dependent operations of the state machine.

3.10.2 Function Documentation

3.10.2.1 timer_is_timer_expired()

```
bool timer_is_timer_expired (
            time_t start_timestamp )
```

Check whether 3 seconds have passed since the timer started.

Parameters

in	*start_time*	Start time of timer.

Returns

1 if the timer is expired, 0 if not.

Definition at line 17 of file timer.c.

3.10.2.2 timer_start_timer()

```
time_t timer_start_timer ( )
```

Starts a fictious timer by returning timestamp of the current time of the system.

Returns

the start time.

Definition at line 12 of file timer.c.

Index