

BFIT

Generated by Doxygen 1.9.1

1 BFIT	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	7
4.1 inventory Struct Reference	7
4.1.1 Detailed Description	8
4.1.2 Member Data Documentation	8
4.1.2.1 number_of_products_stocked	8
4.1.2.2 products_in_inventory	8
4.1.2.3 room_number	8
4.2 product Struct Reference	8
4.2.1 Detailed Description	9
4.2.2 Member Data Documentation	9
4.2.2.1 beverage_variant	9
4.2.2.2 name	9
4.2.2.3 price	9
4.2.2.4 weight	10
4.3 products_stocked Struct Reference	10
4.3.1 Detailed Description	10
4.3.2 Member Data Documentation	11
4.3.2.1 beverage	11
4.3.2.2 current_quantity	11
4.3.2.3 original_quantity	11
4.4 User Struct Reference	11
4.4.1 Detailed Description	11
4.4.2 Member Data Documentation	12
4.4.2.1 balance	12
4.4.2.2 roomNumber	12
4.4.2.3 uid	12
5 File Documentation	13
5.1 include/admin_html.h File Reference	13
5.1.1 Detailed Description	13
5.1.2 Variable Documentation	14
5.1.2.1 PROGMEM	14
5.2 include/buzzer.h File Reference	14
5.2.1 Detailed Description	15
5.2.2 Macro Definition Documentation	15
5.2.2.1 BUZZER_H	15

5.2.3 Function Documentation	15
5.2.3.1 play_lock()	16
5.2.3.2 play_unlock()	16
5.2.3.3 play_warning()	16
5.3 include/fridge_state.h File Reference	16
5.3.1 Detailed Description	17
5.3.2 Variable Documentation	18
5.3.2.1 fridge	18
5.4 include/graph_data.h File Reference	18
5.4.1 Macro Definition Documentation	19
5.4.1.1 ROOM_COUNT	19
5.4.2 Function Documentation	19
5.4.2.1 graph_add_to_room_clasic()	19
5.4.2.2 graph_add_to_room_green()	20
5.4.2.3 print_graph_arrays()	20
5.4.3 Variable Documentation	20
5.4.3.1 classicHeight	20
5.4.3.2 greenHeight	20
5.5 include/index_html.h File Reference	21
5.5.1 Detailed Description	21
5.5.2 Variable Documentation	22
5.5.2.1 PROGMEM	22
5.6 include/init_users_and_sale.h File Reference	22
5.6.1 Detailed Description	23
5.6.2 Macro Definition Documentation	23
5.6.2.1 number_of_users	23
5.6.3 Function Documentation	23
5.6.3.1 init_users_and_products()	24
5.6.3.2 perform_sale()	24
5.7 include/inventory.h File Reference	24
5.7.1 Detailed Description	26
5.7.2 Macro Definition Documentation	26
5.7.2.1 INVENTORY_CAPACITY	26
5.7.3 Enumeration Type Documentation	26
5.7.3.1 beverage_type	26
5.7.4 Function Documentation	26
5.7.4.1 inventory_add_beverage()	26
5.7.4.2 inventory_add_product()	27
5.7.4.3 inventory_init()	27
5.7.4.4 inventory_make_product()	28
5.7.4.5 inventory_print()	28
5.7.4.6 inventory_remove_beverage()	28

5.7.4.7 inventory_remove_product()	29
5.8 include/lock_ctrl.h File Reference	29
5.8.1 Detailed Description	31
5.8.2 Macro Definition Documentation	31
5.8.2.1 CLOSED_THRESHOLD	31
5.8.2.2 OPEN_THRESHOLD	31
5.8.2.3 SERVO_PIN	31
5.8.3 Function Documentation	31
5.8.3.1 is_box_closed()	32
5.8.3.2 lock_ctrl_init()	32
5.8.3.3 lock_door()	32
5.8.3.4 play_close()	32
5.8.3.5 play_open()	32
5.8.3.6 unlock_door()	33
5.9 include/login_html.h File Reference	33
5.9.1 Detailed Description	33
5.9.2 Variable Documentation	33
5.9.2.1 PROGMEM	34
5.10 include/rfid_access.h File Reference	34
5.10.1 Detailed Description	36
5.10.2 Macro Definition Documentation	36
5.10.2.1 MAX_ROOMS	36
5.10.2.2 RST_PIN	36
5.10.2.3 SS_PIN	37
5.10.2.4 UID_LENGTH	37
5.10.3 Enumeration Type Documentation	37
5.10.3.1 RFIDcommand	37
5.10.4 Function Documentation	37
5.10.4.1 add_user()	37
5.10.4.2 check_command()	38
5.10.4.3 compare_UID()	38
5.10.4.4 count_rooms()	38
5.10.4.5 display_commands()	39
5.10.4.6 display_commands_um()	39
5.10.4.7 find_empty_index()	39
5.10.4.8 get_users_db()	39
5.10.4.9 print_all_users()	41
5.10.4.10 print_single_user()	41
5.10.4.11 print_uid()	41
5.10.4.12 read_confirmation()	42
5.10.4.13 read_integer()	42
5.10.4.14 read_RFID_tag()	42

5.10.4.15 remove_user()	42
5.10.4.16 rfid_get_last_uid()	43
5.10.4.17 rfid_set_last_uid()	43
5.10.4.18 setup_RFID_reader()	43
5.10.4.19 user_management()	44
5.10.4.20 validate_rfid()	44
5.10.5 Variable Documentation	44
5.10.5.1 userCount	44
5.10.5.2 users	45
5.11 include/sale_html.h File Reference	45
5.11.1 Detailed Description	46
5.11.2 Function Documentation	46
5.11.2.1 send_sale_html_graph()	46
5.11.2.2 send_sale_html_page()	46
5.11.3 Variable Documentation	47
5.11.3.1 PROGMEM	47
5.12 include/style_css.h File Reference	48
5.12.1 Detailed Description	48
5.12.2 Variable Documentation	48
5.12.2.1 PROGMEM	48
5.13 include/weight_scale.h File Reference	49
5.13.1 Detailed Description	50
5.13.2 Macro Definition Documentation	50
5.13.2.1 BEER_WEIGHT	50
5.13.2.2 HX711_DOUT	50
5.13.2.3 HX711_SCK	51
5.13.2.4 SCALE_DEFAULT_SETTLE_TIME_MS	51
5.13.2.5 SCALE_TOL	51
5.13.3 Function Documentation	51
5.13.3.1 get_beer_cans_taken()	51
5.13.3.2 get_weight()	51
5.13.3.3 get_weight_reference()	52
5.13.3.4 reset_weight_reference()	52
5.13.3.5 set_weight_reference()	52
5.13.3.6 setup_scale()	52
5.13.3.7 tare_complete()	53
5.13.3.8 tare_scale()	53
5.13.3.9 update_scale()	53
5.13.3.10 weight_reference_is_set()	54
5.13.4 Variable Documentation	54
5.13.4.1 scale	54
5.14 README.md File Reference	54

5.15 src/buzzer.cpp File Reference	54
5.15.1 Function Documentation	55
5.15.1.1 play_lock()	55
5.15.1.2 play_unlock()	55
5.15.1.3 play_warning()	55
5.15.2 Variable Documentation	56
5.15.2.1 BUZZERPIN	56
5.15.2.2 HIGH_TONE	56
5.15.2.3 LOW_TONE	56
5.15.2.4 TONE_LENGTH	56
5.16 src/database_management.cpp File Reference	57
5.16.1 Detailed Description	58
5.16.2 Function Documentation	58
5.16.2.1 count_rooms()	58
5.16.2.2 find_empty_index()	58
5.16.2.3 get_users_db()	59
5.16.2.4 print_all_users()	59
5.16.2.5 print_single_user()	59
5.16.2.6 print_uid()	59
5.16.2.7 read_confirmation()	60
5.16.2.8 read_integer()	60
5.16.2.9 remove_user()	60
5.16.2.10 user_management()	60
5.17 src/fridge_state.cpp File Reference	61
5.17.1 Detailed Description	61
5.17.2 Variable Documentation	61
5.17.2.1 fridge	62
5.18 src/graph_data.cpp File Reference	62
5.18.1 Function Documentation	62
5.18.1.1 graph_add_to_room_clasic()	62
5.18.1.2 graph_add_to_room_green()	63
5.18.2 Variable Documentation	63
5.18.2.1 classicHeight	63
5.18.2.2 greenHeight	63
5.19 src/init_users_and_sale.cpp File Reference	64
5.19.1 Function Documentation	64
5.19.1.1 init_users_and_products()	64
5.19.1.2 perform_sale()	64
5.19.1.3 read_current_weight_blocking()	65
5.20 src/inventory.cpp File Reference	65
5.20.1 Detailed Description	66
5.20.2 Function Documentation	66

5.20.2.1 inventory_add_beverage()	66
5.20.2.2 inventory_add_product()	66
5.20.2.3 inventory_init()	67
5.20.2.4 inventory_make_product()	67
5.20.2.5 inventory_print()	68
5.20.2.6 inventory_remove_beverage()	68
5.20.2.7 inventory_remove_product()	68
5.21 src/lock_ctrl.cpp File Reference	69
5.21.1 Detailed Description	70
5.21.2 Function Documentation	70
5.21.2.1 is_box_closed()	70
5.21.2.2 lock_ctrl_init()	70
5.21.2.3 lock_door()	71
5.21.2.4 play_close()	71
5.21.2.5 play_open()	71
5.21.2.6 unlock_door()	71
5.21.3 Variable Documentation	71
5.21.3.1 boxClosed	71
5.21.3.2 BUZZER	72
5.21.3.3 HIGH_TONE	72
5.21.3.4 LOCK_POS	72
5.21.3.5 lockServo	72
5.21.3.6 LOW_TONE	72
5.21.3.7 TONE_LENGTH	72
5.21.3.8 UNLOCK_POS	73
5.22 src/main.cpp File Reference	73
5.22.1 Detailed Description	74
5.22.2 Function Documentation	74
5.22.2.1 connect_wifi_and_start_mdns()	74
5.22.2.2 loop()	74
5.22.2.3 print_graph_arrays()	74
5.22.2.4 rfid()	75
5.22.2.5 server()	75
5.22.2.6 setup()	75
5.22.2.7 setup_inventory_and_scale()	75
5.22.2.8 setup_rfid_and_lock()	75
5.22.2.9 setup_web_routes()	75
5.22.3 Variable Documentation	75
5.22.3.1 activeCommand	75
5.22.3.2 CAL_FACTOR	76
5.22.3.3 classicHeight	76
5.22.3.4 demo_beer	76

5.22.3.5 doorCloseTimer	76
5.22.3.6 doorUnlocked	76
5.22.3.7 greenHeight	76
5.22.3.8 START_BEER_QTY	77
5.22.3.9 WIFI_PASS	77
5.22.3.10 WIFI_SSID	77
5.23 src/rfid_access.cpp File Reference	77
5.23.1 Detailed Description	78
5.23.2 Function Documentation	78
5.23.2.1 add_user()	78
5.23.2.2 check_command()	79
5.23.2.3 compare_UID()	79
5.23.2.4 display_commands()	79
5.23.2.5 display_commands_um()	80
5.23.2.6 read_RFID_tag()	80
5.23.2.7 rfid_get_last_uid()	80
5.23.2.8 rfid_set_last_uid()	81
5.23.2.9 setup_RFID_reader()	81
5.23.2.10 validate_rfid()	81
5.23.3 Variable Documentation	82
5.23.3.1 hasUID	82
5.23.3.2 lastUID	82
5.23.3.3 userCount	82
5.23.3.4 users	82
5.24 src/sale_html.cpp File Reference	82
5.24.1 Detailed Description	83
5.24.2 Function Documentation	83
5.24.2.1 send_sale_html_graph()	83
5.24.2.2 send_sale_html_page()	84
5.24.3 Variable Documentation	84
5.24.3.1 PROGMEM	84
5.25 src/weight_scale.cpp File Reference	85
5.25.1 Detailed Description	86
5.25.2 Function Documentation	86
5.25.2.1 get_beer_cans_taken()	86
5.25.2.2 get_weight()	87
5.25.2.3 get_weight_reference()	87
5.25.2.4 reset_weight_reference()	87
5.25.2.5 scale()	87
5.25.2.6 set_weight_reference()	87
5.25.2.7 setup_scale()	88
5.25.2.8 tare_complete()	88

5.25.2.9 tare_scale()	88
5.25.2.10 update_scale()	88
5.25.2.11 weight_reference_is_set()	89
5.25.3 Variable Documentation	89
5.25.3.1 g_referenceWeight	89
Index	91

Chapter 1

BFIT

Beer Fridge Inventory Tracking (who takes, how much do they take, current drink information, statistics and so on and so forth.)

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

inventory	Represents a user's beverage inventory	7
product	Describes a single beverage product	8
products_stocked	Tracks inventory quantities for a single product	10
User	User record stored in the RFID user database	11

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

include/ admin_html.h	13
HTML for the admin page	
include/ buzzer.h	14
Implementation of buzzer sound effects	
include/ fridge_state.h	16
File for declaring the fridge inventory, user inventory should also be moved to this file, and the file renamed to reflect the new content	
include/ graph_data.h	18
include/ index_html.h	21
HTML for the start page	
include/ init_users_and_sale.h	22
Function used for declaring the system users and products	
include/ inventory.h	24
Inventory system for tracking the inventory of both the fridge and the individual users	
include/ lock_ctrl.h	29
Door lock control interface using a servo and light sensor	
include/ login_html.h	33
HTML file for the login page	
include/ rfid_access.h	34
include/ sale_html.h	45
Headerfile for displaying the graph of sales on the main page	
include/ style_css.h	48
CSS served at /style.css used for setting the style for the web server	
include/ weight_scale.h	49
src/ buzzer.cpp	54
src/ database_management.cpp	57
Used to read and write non-volatile memory on ESP8266	
src/ fridge_state.cpp	61
src/ graph_data.cpp	62
src/ init_users_and_sale.cpp	64
src/ inventory.cpp	65
Functions responsible for keeping track of the fridge inventory	
src/ lock_ctrl.cpp	69
src/ main.cpp	73
Combined: Web server + Graph + Scale + RFID access + Lock control	
src/ rfid_access.cpp	77
src/ sale_html.cpp	82
src/ weight_scale.cpp	85

Chapter 4

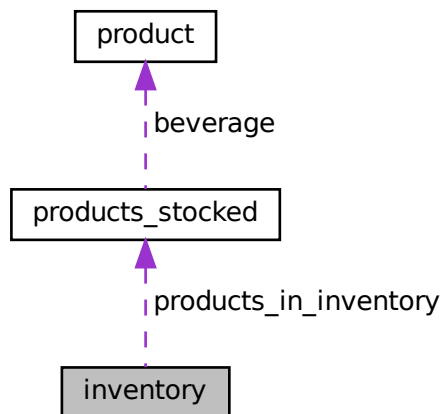
Class Documentation

4.1 inventory Struct Reference

Represents a user's beverage inventory.

```
#include <inventory.h>
```

Collaboration diagram for inventory:



Public Attributes

- [products_stocked](#) [products_in_inventory](#) [[INVENTORY_CAPACITY](#)]
Products currently in inventory.
- [uint8_t](#) [number_of_products_stocked](#)
Number of valid entries in products_in_inventory.
- [uint8_t](#) [room_number](#)
Room number associated with the inventory (currently unused)

4.1.1 Detailed Description

Represents a user's beverage inventory.

4.1.2 Member Data Documentation

4.1.2.1 number_of_products_stocked

```
uint8_t inventory::number_of_products_stocked
```

Number of valid entries in products_in_inventory.

4.1.2.2 products_in_inventory

```
products_stocked inventory::products_in_inventory[INVENTORY_CAPACITY]
```

Products currently in inventory.

4.1.2.3 room_number

```
uint8_t inventory::room_number
```

Room number associated with the inventory (currently unused)

FRID of the user owning the inventory (not implemented)

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

- include/[inventory.h](#)

4.2 product Struct Reference

Describes a single beverage product.

```
#include <inventory.h>
```

Public Attributes

- char `name` [20]
Display name of the beverage.
- `beverage_type` `beverage_variant`
Beverage type.
- uint16_t `weight`
Weight of the beverage (e.g.
- uint8_t `price`
Price of the beverage.

4.2.1 Detailed Description

Describes a single beverage product.

4.2.2 Member Data Documentation

4.2.2.1 `beverage_variant`

`beverage_type` `product::beverage_variant`

Beverage type.

4.2.2.2 `name`

`char` `product::name` [20]

Display name of the beverage.

4.2.2.3 `price`

`uint8_t` `product::price`

Price of the beverage.

4.2.2.4 weight

```
uint16_t product::weight
```

Weight of the beverage (e.g.

grams)

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

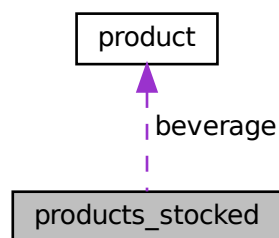
- [include/inventory.h](#)

4.3 products_stocked Struct Reference

Tracks inventory quantities for a single product.

```
#include <inventory.h>
```

Collaboration diagram for products_stocked:



Public Attributes

- [product beverage](#)
Product being tracked.
- `uint8_t` [original_quantity](#)
Initial stock quantity.
- `uint8_t` [current_quantity](#)
Current stock quantity.

4.3.1 Detailed Description

Tracks inventory quantities for a single product.

4.3.2 Member Data Documentation

4.3.2.1 beverage

`product products_stocked::beverage`

Product being tracked.

4.3.2.2 current_quantity

`uint8_t products_stocked::current_quantity`

Current stock quantity.

4.3.2.3 original_quantity

`uint8_t products_stocked::original_quantity`

Initial stock quantity.

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

- `include/inventory.h`

4.4 User Struct Reference

`User` record stored in the RFID user database.

```
#include <rfid_access.h>
```

Public Attributes

- `byte uid [UID_LENGTH]`
- `int roomNumber`
- `int balance`

4.4.1 Detailed Description

`User` record stored in the RFID user database.

4.4.2 Member Data Documentation

4.4.2.1 balance

```
int User::balance
```

4.4.2.2 roomNumber

```
int User::roomNumber
```

4.4.2.3 uid

```
byte User::uid[UID_LENGTH]
```

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

- [include/rfid_access.h](#)

Chapter 5

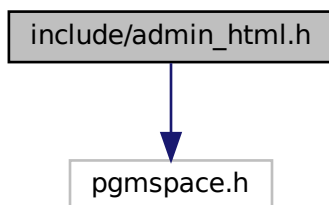
File Documentation

5.1 include/admin_html.h File Reference

HTML for the admin page.

```
#include <pgmspace.h>
```

Include dependency graph for admin_html.h:



Variables

- `const char ADMIN_HTML[]` [PROGMEM](#)
HTML content for the admin web page.

5.1.1 Detailed Description

HTML for the admin page.

Authors

Baldur G. Toftegaard

5.1.2 Variable Documentation

5.1.2.1 PROGMEM

```
const char ADMIN_HTML [ ] PROGMEM
```

Initial value:

```
= R"rawliteral(  
  <!DOCTYPE html>  
  <html>  
    <head>  
      <meta charset="utf-8">  
      <title>Beer fridge online services</title>  
      <link rel="stylesheet" href="/style.css">  
    </head>  
    <body>  
      <div class="topbar">  
        <div class="left">  
          Start -> Admin  
        </div>  
        <div class="right">  
          <a href="/">Log Out</a>  
        </div>  
      </div>  
      <p> Verry inportant stuff goes here! </p>  
    </body>  
  </html>  
)rawliteral"
```

HTML content for the admin web page.

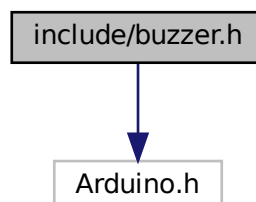
Opening container for the sales graph.

5.2 include/buzzer.h File Reference

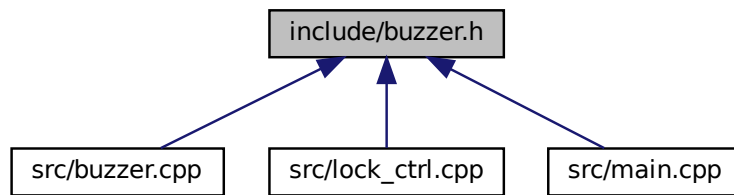
Implementation of buzzer sound effects.

```
#include <Arduino.h>
```

Include dependency graph for buzzer.h:



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



Macros

- `#define BUZZER_H`

Functions

- void `play_warning` (unsigned long t)
Plays short sound when door is closed and about to be locked.
- void `play_unlock` ()
Plays an ascending tone, when the door unlocks.
- void `play_lock` ()
Play a descending tone, when the door locks.

5.2.1 Detailed Description

Implementation of buzzer sound effects.

Author

Anssi Sohlman

5.2.2 Macro Definition Documentation

5.2.2.1 BUZZER_H

```
#define BUZZER_H
```

5.2.3 Function Documentation

5.2.3.1 play_lock()

```
void play_lock ( )
```

Play a descending tone, when the door locks.

5.2.3.2 play_unlock()

```
void play_unlock ( )
```

Plays an ascending tone, when the door unlocks.

5.2.3.3 play_warning()

```
void play_warning (
    unsigned long t )
```

Plays short sound when door is closed and about to be locked.

Parameters

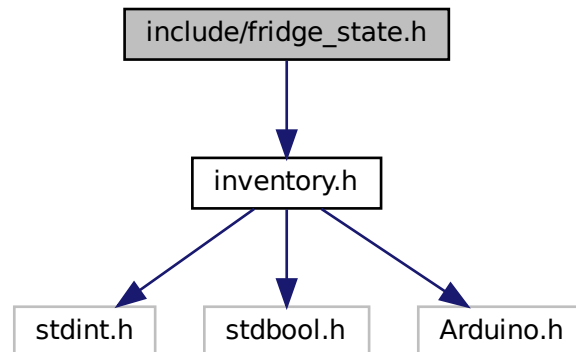
<i>t</i>	
----------	--

5.3 include/fridge_state.h File Reference

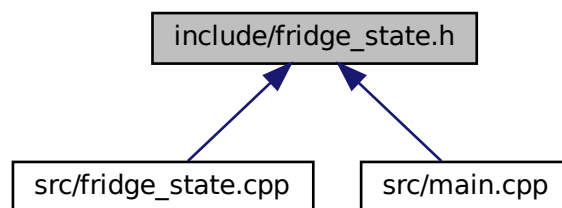
File for declaring the fridge inventory, user inventory should also be moved to this file, and the file renamed to reflect the new content.

```
#include "inventory.h"
```

Include dependency graph for fridge_state.h:



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



Variables

- [inventory fridge](#)

Used to set up the fridge inventory.

5.3.1 Detailed Description

File for declaring the fridge inventory, user inventory should also be moved to this file, and the file renamed to reflect the new content.

Author

Baldur G. Toftegaard

5.3.2 Variable Documentation

5.3.2.1 fridge

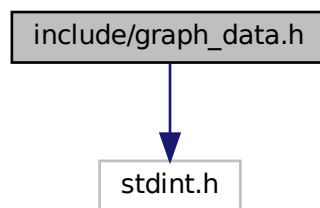
```
inventory fridge [extern]
```

Used to set up the fridge inventory.

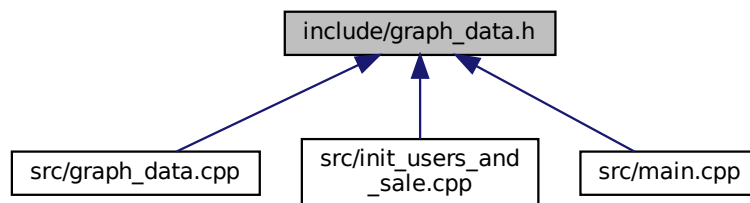
5.4 include/graph_data.h File Reference

```
#include <stdint.h>
```

Include dependency graph for graph_data.h:



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



Macros

- `#define ROOM_COUNT 18`
Number of rooms supported by the sales graph.

Functions

- void [graph_add_to_room_green](#) (uint8_t roomNumber, int delta)
Adds a value to the green sales bar of a given room.
- void [graph_add_to_room_clasic](#) (uint8_t roomNumber, int delta)
Adds a value to the classic sales bar of a given room.
- void [print_graph_arrays](#) ()
Prints the current graph height arrays.

Variables

- int [greenHeight](#) [[ROOM_COUNT](#)]
Height values for green product sales per room.
- int [classicHeight](#) [[ROOM_COUNT](#)]
Height values for classic product sales per room.

5.4.1 Macro Definition Documentation

5.4.1.1 ROOM_COUNT

```
#define ROOM_COUNT 18
```

Number of rooms supported by the sales graph.

5.4.2 Function Documentation

5.4.2.1 graph_add_to_room_clasic()

```
void graph_add_to_room_clasic (  
    uint8_t roomNumber,  
    int delta )
```

Adds a value to the classic sales bar of a given room.

Parameters

<i>roomNumber</i>	
<i>delta</i>	

5.4.2.2 graph_add_to_room_green()

```
void graph_add_to_room_green (
    uint8_t roomNumber,
    int delta )
```

Adds a value to the green sales bar of a given room.

Parameters

<i>roomNumber</i>	
<i>delta</i>	

5.4.2.3 print_graph_arrays()

```
void print_graph_arrays ( )
```

Prints the current graph height arrays.

5.4.3 Variable Documentation

5.4.3.1 classicHeight

```
int classicHeight[ROOM_COUNT] [extern]
```

Height values for classic product sales per room.

Height values for classic product sales per room.

Indexed by room number. Used by sale_html and the /saleHeights endpoint.

5.4.3.2 greenHeight

```
int greenHeight[ROOM_COUNT] [extern]
```

Height values for green product sales per room.

Height values for green product sales per room.

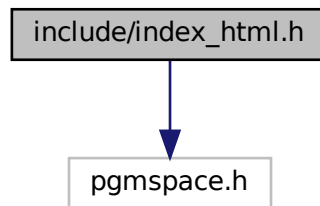
Indexed by room number. Used by sale_html and the /saleHeights endpoint.

5.5 include/index_html.h File Reference

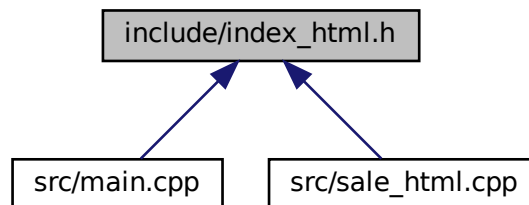
HTML for the start page.

```
#include <pgmspace.h>
```

Include dependency graph for index_html.h:



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



Variables

- `const char INDEX_HTML_HEAD[]` [PROGMEM](#)
String used for the HTML header.

5.5.1 Detailed Description

HTML for the start page.

Author

Baldur G. Toftegaard

5.5.2 Variable Documentation

5.5.2.1 PROGMEM

```
const char INDEX_HTML_FOOT [ ] PROGMEM
```

Initial value:

```
= R"rawliteral(  
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="utf-8">  
    <title>Beer fridge online services</title>  
    <link rel="stylesheet" href="/style.css">  
  </head>  
  <body>  
    <div class="topbar">  
      <div class="left">Welcome!</div>  
      <div class="right"><a href="/login">Login</a></div>  
    </div>  
)rawliteral"
```

String used for the HTML header.

Opening container for the sales graph.

String used for the HTML footer.

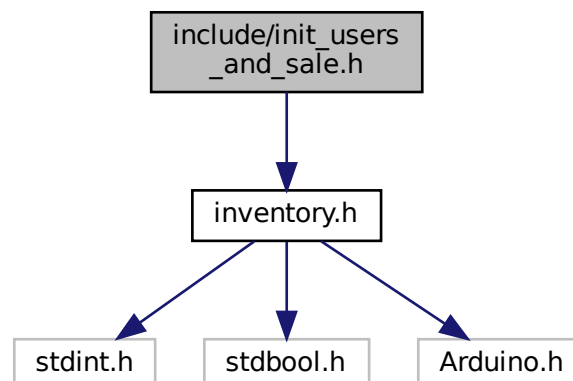
This is responsible for updating the graphs.

5.6 include/init_users_and_sale.h File Reference

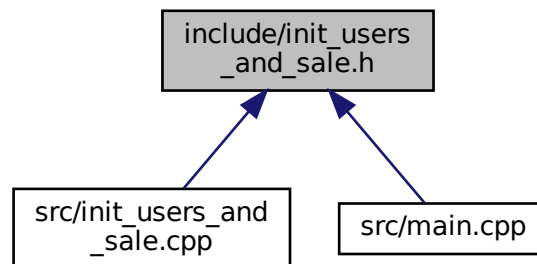
Function used for declaring the system users and products.

```
#include "inventory.h"
```

Include dependency graph for init_users_and_sale.h:



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



Macros

- `#define` `number_of_users` 18

Functions

- void `init_users_and_products` ()
Old function for initializing users and products.
- void `perform_sale` (`inventory` *fridge_inventory)
Performs a sale between a user and the fridge.

5.6.1 Detailed Description

Function used for declaring the system users and products.

Author

Baldur G. Toftegaard

5.6.2 Macro Definition Documentation

5.6.2.1 number_of_users

```
#define number_of_users 18
```

5.6.3 Function Documentation

5.6.3.1 init_users_and_products()

```
void init_users_and_products ( )
```

Old function for initializing users and products.

Old function for initializing users and products.

5.6.3.2 perform_sale()

```
void perform_sale (
    inventory * fridge_inventory )
```

Performs a sale between a user and the fridge.

Parameters

<i>weight</i>	
<i>user_id</i>	
<i>fridge_inventory</i>	

Parameters

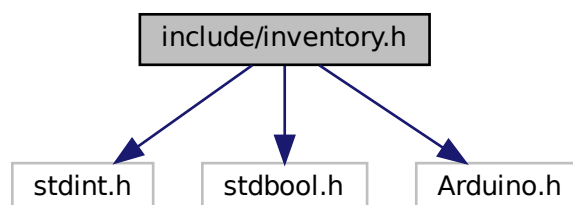
<i>fridge_inventory</i>	Inventory that the sale should remove item from
-------------------------	---

5.7 include/inventory.h File Reference

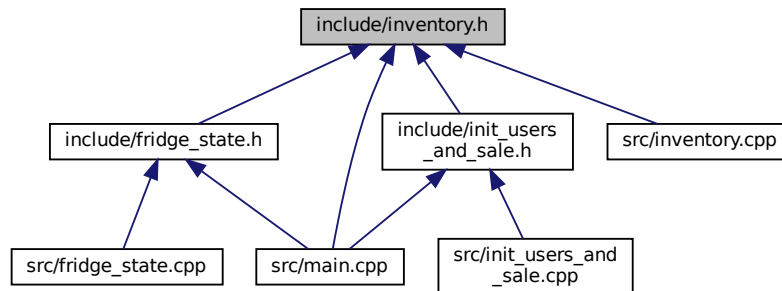
Inventory system for tracking the inventory of both the fridge and the individual users.

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

Include dependency graph for inventory.h:



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



Classes

- struct [product](#)
Describes a single beverage product.
- struct [products_stocked](#)
Tracks inventory quantities for a single product.
- struct [inventory](#)
Represents a user's beverage inventory.

Macros

- `#define` [INVENTORY_CAPACITY](#) 6
Maximum number of products the inventory can hold.

Enumerations

- enum [beverage_type](#) {
 [beer](#) , [cider](#) , [soda](#) , [limfjords_porter](#) ,
 [other](#) }
Types of beverages supported by the system.

Functions

- void [inventory_init](#) ([inventory](#) *[inventory](#))
Initializes an inventory structure.
- [product](#) [inventory_make_product](#) (const char *name, [beverage_type](#) type, uint16_t weight, uint8_t price)
Creates a new product instance.
- bool [inventory_add_product](#) ([inventory](#) *[inventory](#), [product](#) product, uint16_t quantity)
Adds a new product to the inventory.
- bool [inventory_remove_product](#) ([inventory](#) *[inventory](#), [product](#) beverage)
Removes a product entirely from the inventory.
- bool [inventory_add_beverage](#) ([inventory](#) *[inventory](#), [product](#) beverage, uint16_t amount)
Increases the quantity of a beverage in the inventory.
- bool [inventory_remove_beverage](#) ([inventory](#) *[inventory](#), [product](#) beverage, uint8_t amount)
Function for removing from the amount of a beverage in an inventory.
- void [inventory_print](#) ([inventory](#) *[inventory](#))
Function to print a users inventory.

5.7.1 Detailed Description

Inventory system for tracking the inventory of both the fridge and the individual users.

Author

Baldur G. Toftegaard

5.7.2 Macro Definition Documentation

5.7.2.1 INVENTORY_CAPACITY

```
#define INVENTORY_CAPACITY 6
```

Maximum number of products the inventory can hold.

5.7.3 Enumeration Type Documentation

5.7.3.1 beverage_type

```
enum beverage_type
```

Types of beverages supported by the system.

Enumerator

beer	Beer.
cider	Cider.
soda	Soda.
limfjords_porter	Limfjords porter.
other	Other.

5.7.4 Function Documentation

5.7.4.1 inventory_add_beverage()

```
bool inventory_add_beverage (  
    inventory * inventory,
```

```
product beverage,  
uint16_t amount )
```

Increases the quantity of a beverage in the inventory.

Parameters

<i>inventory</i>	Inventory to modify.
<i>beverage</i>	Beverage to update.
<i>amount</i>	Quantity to add.

Returns

true Quantity updated successfully.

false Product not found or overflow.

5.7.4.2 inventory_add_product()

```
bool inventory_add_product (  
    inventory * inventory,  
    product product,  
    uint16_t quantity )
```

Adds a new product to the inventory.

Parameters

<i>inventory</i>	Inventory to add the product to.
<i>product</i>	Product to add.
<i>quantity</i>	Initial quantity of the product.

Returns

true Product successfully added.

false Error occurred (inventory full or duplicate product).

5.7.4.3 inventory_init()

```
void inventory_init (  
    inventory * inventory )
```

Initializes an inventory structure.

Clears internal state and prepares the inventory for use.

Parameters

<i>inventory</i>	Pointer to inventory instance to initialize.
------------------	--

5.7.4.4 inventory_make_product()

```
product inventory_make_product (
    const char * name,
    beverage_type type,
    uint16_t weight,
    uint8_t price )
```

Creates a new product instance.

Parameters

<i>name</i>	Display name of the product.
<i>type</i>	Beverage type.
<i>weight</i>	Weight of the product.
<i>price</i>	Price of the product.

Returns

Initialized product instance.

5.7.4.5 inventory_print()

```
void inventory_print (
    inventory * inventory )
```

Function to print a users inventory.

Parameters

<i>inventory</i>	Inventory to print.
------------------	---------------------

5.7.4.6 inventory_remove_beverage()

```
bool inventory_remove_beverage (
    inventory * inventory,
```

```
product beverage,  
uint8_t amount )
```

Function for removing from the amount of a beverage in an inventory.

Parameters

<i>inventory</i>	Inventory to modify.
<i>beverage</i>	Beverage to update.
<i>amount</i>	Quantity to remove.

Returns

true Quantity updated successfully.
false Product not found or insufficient stock.

5.7.4.7 inventory_remove_product()

```
bool inventory_remove_product (  
    inventory * inventory,  
    product beverage )
```

Removes a product entirely from the inventory.

Parameters

<i>inventory</i>	Inventory to remove the product from.
<i>beverage</i>	Product to remove.

Returns

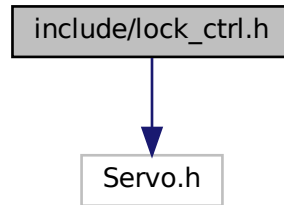
true Product removed successfully.
false Product not found.

5.8 include/lock_ctrl.h File Reference

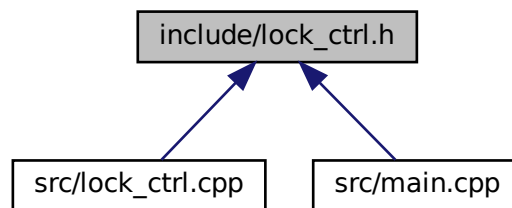
Door lock control interface using a servo and light sensor.

```
#include <Servo.h>
```

Include dependency graph for lock_ctrl.h:



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



Macros

- #define `SERVO_PIN` 16
GPIO pin connected to the servo motor.
- #define `CLOSED_THRESHOLD` 70
Light threshold indicating a closed box.
- #define `OPEN_THRESHOLD` 100
Light threshold indicating an open box.

Functions

- void `lock_ctrl_init` ()
Initializes the lock control module.
- void `lock_door` ()
Locks the door using the servo.
- void `unlock_door` ()
Unlocks the door using the servo.
- bool `is_box_closed` ()

- Checks whether the box is closed.*
 - void `play_open` ()
Plays the sound effect for door opening.
 - void `play_close` ()
Play sound effect when the door closes.

5.8.1 Detailed Description

Door lock control interface using a servo and light sensor.

Author

Amal Araweelo Almis

5.8.2 Macro Definition Documentation

5.8.2.1 CLOSED_THRESHOLD

```
#define CLOSED_THRESHOLD 70
```

Light threshold indicating a closed box.

5.8.2.2 OPEN_THRESHOLD

```
#define OPEN_THRESHOLD 100
```

Light threshold indicating an open box.

5.8.2.3 SERVO_PIN

```
#define SERVO_PIN 16
```

GPIO pin connected to the servo motor.

5.8.3 Function Documentation

5.8.3.1 is_box_closed()

```
bool is_box_closed ( )
```

Checks whether the box is closed.

Uses the light sensor to determine door state.

Returns

true Box is closed.

false Box is open.

5.8.3.2 lock_ctrl_init()

```
void lock_ctrl_init ( )
```

Initializes the lock control module.

Must be called once during system startup before any other lock control functions are used.

5.8.3.3 lock_door()

```
void lock_door ( )
```

Locks the door using the servo.

5.8.3.4 play_close()

```
void play_close ( )
```

Play sound effect when the door closes.

5.8.3.5 play_open()

```
void play_open ( )
```

Plays the sound effect for door opening.

5.8.3.6 unlock_door()

```
void unlock_door ( )
```

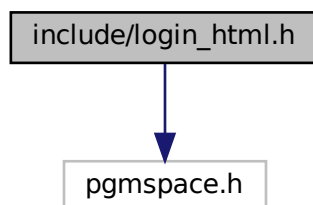
Unlocks the door using the servo.

5.9 include/login_html.h File Reference

HTML file for the login page.

```
#include <pgmspace.h>
```

Include dependency graph for login_html.h:



Variables

- `const char LOGIN_HTML[]` [PROGMEM](#)
HTML for the login page.

5.9.1 Detailed Description

HTML file for the login page.

Authors

Baldur G. Toftegaard

5.9.2 Variable Documentation

5.9.2.1 PROGMEM

```
const char LOGIN_HTML [ ] PROGMEM
```

HTML for the login page.

Opening container for the sales graph.

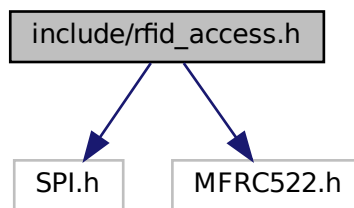
Interpreted as a string by the compiler.

5.10 include/rfid_access.h File Reference

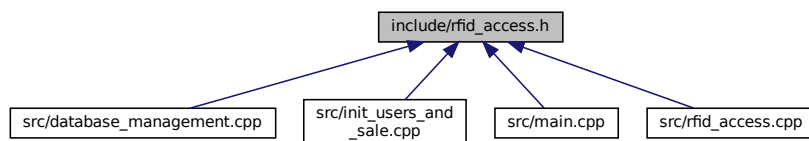
```
#include <SPI.h>
```

```
#include <MFRC522.h>
```

Include dependency graph for rfid_access.h:



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



Classes

- struct [User](#)

[User](#) record stored in the RFID user database.

Macros

- `#define SS_PIN 15`
GPIO pin connected to SS.
- `#define RST_PIN 0`
GPIO pin connected to RST.
- `#define MAX_ROOMS 17`
Maximum number of rooms supported by the system.
- `#define UID_LENGTH 4`
Length of an RFID UID in bytes.

Enumerations

- enum `RFIDcommand` {
`CMD_NONE` , `CMD_ADD_USER` , `CMD_OPEN` , `CMD_LOCK` ,
`CMD_REMOVE_USER` , `CMD_CONFIRM` , `CMD_PRINT` }
Supported serial commands for the RFID management interface.

Functions

- `RFIDcommand check_command` (void)
brief Reads a command from the serial interface and maps it to an RFIDcommand.
- void `setup_RFID_reader` (MFRC522 &rfid)
Initializes SPI and the MFRC522 RFID reader.
- bool `add_user` (MFRC522 &rfid)
Adds a new user by reading room number and scanning an RFID tag.
- bool `remove_user` ()
Removes a user from the database.
- bool `compare_UID` (byte *uid1, byte *uid2)
Compares two RFID UIDs.
- bool `read_RFID_tag` (MFRC522 &rfid, byte *uidBuffer)
Reads an RFID tag UID from the MFRC522 reader.
- void `display_commands` (void)
Prints the available serial commands.
- void `display_commands_um` ()
Prints the available serial commands for user-management mode.
- void `get_users_db` (User *ptr)
Copies the current user database to a provided buffer.
- void `user_management` (RFIDcommand cmd, User *ptr, MFRC522 &rfid)
Executes user-management actions based on the provided command.
- bool `validate_rfid` (MFRC522 myRFID)
Validates an RFID tag against the registered user database.
- void `print_single_user` (User *ptr, int idx)
brief Prints a single user entry to the serial interface.
- void `print_all_users` (User *ptr)
Prints all users in the database to the serial interface.
- void `print_uid` (byte *ptr)
Prints a UID buffer to the serial interface.
- int `read_integer` ()
Reads an integer from the serial interface.

- bool `read_confirmation` ()
Reads a confirmation input from the serial interface.
- int `find_empty_index` (User *ptr)
brief Finds an empty slot in the user database.
- int `count_rooms` (User *ptr)
brief Counts the number of occupied user entries in the database.
- void `rfid_set_last_uid` (const byte *uidIn)
Function for storing the last used RFID.
- bool `rfid_get_last_uid` (byte *uidOut)
Function for restoring the last used RFID.

Variables

- User `users` [MAX_ROOMS]
Global user database array.
- int `userCount`
Number of currently registered users.

5.10.1 Detailed Description

Author

Amal Araweelo Almis
Baldur G. Toftegaard
Anssi Sohlman

5.10.2 Macro Definition Documentation

5.10.2.1 MAX_ROOMS

```
#define MAX_ROOMS 17
```

Maximum number of rooms supported by the system.

5.10.2.2 RST_PIN

```
#define RST_PIN 0
```

GPIO pin connected to RST.

5.10.2.3 SS_PIN

```
#define SS_PIN 15
```

GPIO pin connected to SS.

5.10.2.4 UID_LENGTH

```
#define UID_LENGTH 4
```

Length of an RFID UID in bytes.

5.10.3 Enumeration Type Documentation

5.10.3.1 RFIDcommand

```
enum RFIDcommand
```

Supported serial commands for the RFID management interface.

Enumerator

CMD_NONE	
CMD_ADD_USER	
CMD_OPEN	
CMD_LOCK	
CMD_REMOVE_USER	
CMD_CONFIRM	
CMD_PRINT	

5.10.4 Function Documentation

5.10.4.1 add_user()

```
bool add_user (
    MFRC522 & rfid )
```

Adds a new user by reading room number and scanning an RFID tag.

Parameters

<i>rfid</i>	
-------------	--

Returns

true
false

5.10.4.2 check_command()

```
RFIDcommand check_command (  
    void )
```

brief Reads a command from the serial interface and maps it to an RFIDcommand.

Returns

RFIDcommand

5.10.4.3 compare_UID()

```
bool compare_UID (  
    byte * uid1,  
    byte * uid2 )
```

Compares two RFID UUIDs.

Parameters

<i>uid1</i>	
<i>uid2</i>	

Returns

true
false

5.10.4.4 count_rooms()

```
int count_rooms (  
    User * ptr )
```

brief Counts the number of occupied user entries in the database.

Parameters

<i>ptr</i>	
------------	--

Returns

int

5.10.4.5 display_commands()

```
void display_commands (
    void )
```

Prints the available serial commands.

5.10.4.6 display_commands_um()

```
void display_commands_um ( )
```

Prints the available serial commands for user-management mode.

5.10.4.7 find_empty_index()

```
int find_empty_index (
    User * ptr )
```

brief Finds an empty slot in the user database.

Parameters

<i>ptr</i>	
------------	--

Returns

int

5.10.4.8 get_users_db()

```
void get_users_db (
    User * ptr )
```

Copies the current user database to a provided buffer.

Parameters

<i>ptr</i>	
------------	--

5.10.4.9 print_all_users()

```
void print_all_users (
    User * ptr )
```

Prints all users in the database to the serial interface.

Parameters

<i>ptr</i>	
------------	--

5.10.4.10 print_single_user()

```
void print_single_user (
    User * ptr,
    int idx )
```

brief Prints a single user entry to the serial interface.

Parameters

<i>ptr</i>	
<i>idx</i>	

5.10.4.11 print_uid()

```
void print_uid (
    byte * ptr )
```

Prints a UID buffer to the serial interface.

Parameters

<i>ptr</i>	
------------	--

5.10.4.12 read_confirmation()

```
bool read_confirmation ( )
```

Reads a confirmation input from the serial interface.

Returns

true
false

5.10.4.13 read_integer()

```
int read_integer ( )
```

Reads an integer from the serial interface.

Returns

int

5.10.4.14 read_RFID_tag()

```
bool read_RFID_tag (
    MFRC522 & rfid,
    byte * uidBuffer )
```

Reads an RFID tag UID from the MFRC522 reader.

Parameters

<i>rfid</i>	
<i>uidBuffer</i>	

Returns

true
false

5.10.4.15 remove_user()

```
bool remove_user ( )
```

Removes a user from the database.

Returns

true
false

5.10.4.16 rfid_get_last_uid()

```
bool rfid_get_last_uid (
    byte * uidOut )
```

Function for restoring the last used RFID.

Parameters

<i>uidOut</i>	
---------------	--

Returns

true
false

5.10.4.17 rfid_set_last_uid()

```
void rfid_set_last_uid (
    const byte * uidIn )
```

Function for storing the last used RFID.

Parameters

<i>uidOut</i>	
---------------	--

Returns

true
false

5.10.4.18 setup_RFID_reader()

```
void setup_RFID_reader (
    MFRC522 & rfid )
```

Initializes SPI and the MFRC522 RFID reader.

Parameters

<i>rfid</i>	
-------------	--

5.10.4.19 user_management()

```
void user_management (
    RFIDcommand cmd,
    User * ptr,
    MFRC522 & rfid )
```

Executes user-management actions based on the provided command.

Parameters

<i>cmd</i>	
<i>ptr</i>	
<i>rfid</i>	

5.10.4.20 validate_rfid()

```
bool validate_rfid (
    MFRC522 myRFID )
```

Validates an RFID tag against the registered user database.

Parameters

<i>myRFID</i>	
---------------	--

Returns

true
false

5.10.5 Variable Documentation**5.10.5.1 userCount**

```
int userCount [extern]
```

Number of currently registered users.

5.10.5.2 users

```
User users[MAX_ROOMS] [extern]
```

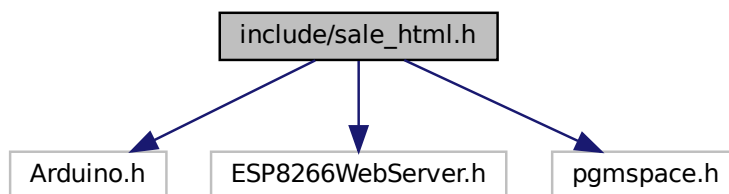
Global user database array.

5.11 include/sale_html.h File Reference

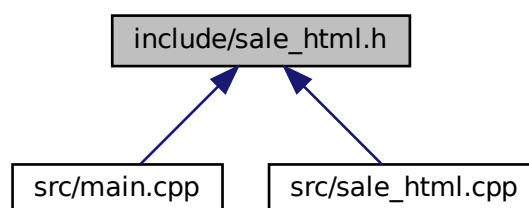
Headerfile for displaying the graph of sales on the main page.

```
#include <Arduino.h>
#include <ESP8266WebServer.h>
#include <pgmspace.h>
```

Include dependency graph for sale_html.h:



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



Functions

- void [send_sale_html_graph](#) (ESP8266WebServer &[server](#), uint8_t room_number, const char *bar_type, int bar_height)
Sends a single sales bar element to the client.
- void [send_sale_html_page](#) (ESP8266WebServer &[server](#), uint8_t room_count, const int *greenHeight, const int *classicHeights)
Sends the complete sales graph page.

Variables

- `const char SALE_BOX_START[]` [PROGMEM](#)
Opening container for the sales graph.

5.11.1 Detailed Description

Headerfile for displaying the graph of sales on the main page.

Authors

Baldur G. Toftegaard

5.11.2 Function Documentation

5.11.2.1 `send_sale_html_graph()`

```
void send_sale_html_graph (
    ESP8266WebServer & server,
    uint8_t room_number,
    const char * bar_type,
    int bar_height )
```

Sends a single sales bar element to the client.

Parameters

<i>server</i>	
<i>room_number</i>	
<i>bar_type</i>	
<i>bar_height</i>	

Parameters

<i>server</i>	The server
<i>room_number</i>	Number of the relevant room
<i>bar_type</i>	The type of the bar graph
<i>bar_height</i>	The height of the bar graph

5.11.2.2 `send_sale_html_page()`

```
void send_sale_html_page (
    ESP8266WebServer & server,
```



```
uint8_t room_count,
const int * greenHeight,
const int * classicHeights )
```

Sends the complete sales graph page.

Parameters

<i>server</i>	
<i>room_count</i>	
<i>greenHeight</i>	
<i>classicHeights</i>	

Parameters

<i>server</i>	The server
<i>room_count</i>	The number of rooms
<i>greenHeight</i>	The hight of the green bar
<i>classicHeights</i>	The hight of the clasic bar (not used in prototype)

5.11.3 Variable Documentation

5.11.3.1 PROGMEM

```
const char SALE_BOX_STOP [ ] PROGMEM [extern]
```

Opening container for the sales graph.

Closing container for the complete sales graph.

Closing container for a single room graph.

Closing fragment for a single sales bar.

HTML fragment defining the height style of a bar.

HTML fragment defining the CSS class type for a bar.

HTML fragment for the room identifier.

Opening container for a single room graph.

Opening container for the sales graph.

Opening container for the sales graph.

String used for the HTML footer.

This is responsible for updating the graphs.

Opening container for the sales graph.

Interpreted as a string by the compiler.

Opening container for the sales graph.

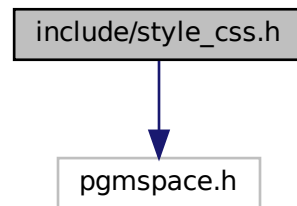
is handled like a string by the compiler

5.12 include/style_css.h File Reference

CSS served at /style.css used for setting the style for the web server.

```
#include <pgmspace.h>
```

Include dependency graph for style_css.h:



Variables

- `const char STYLE_CSS[]` [PROGMEM](#)
String containing the CSS styling of the webserver.

5.12.1 Detailed Description

CSS served at /style.css used for setting the style for the web server.

The css is handled as a string by the compiler.

5.12.2 Variable Documentation

5.12.2.1 PROGMEM

```
const char STYLE_CSS [ ] PROGMEM
```

String containing the CSS styling of the webserver.

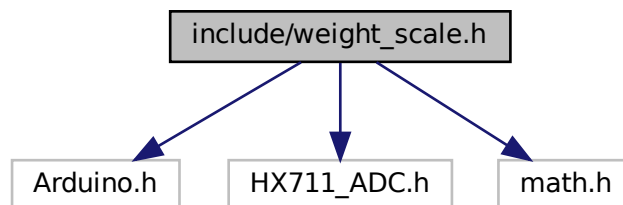
Opening container for the sales graph.

is handled like a string by the compiler

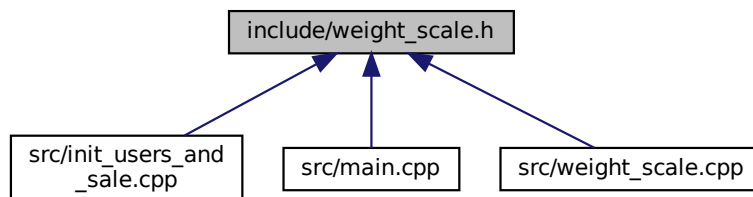
5.13 include/weight_scale.h File Reference

```
#include <Arduino.h>
#include <HX711_ADC.h>
#include <math.h>
```

Include dependency graph for weight_scale.h:



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



Macros

- `#define BEER_WEIGHT 350`
Define the weight of a beer.
- `#define SCALE_TOL 25`
Define the uncertainty of the weight mesurment.
- `#define HX711_DOUT 4`
- `#define HX711_SCK 5`
- `#define SCALE_DEFAULT_SETTLE_TIME_MS 3000`

Functions

- float `get_weight_reference` (void)
Function to get the weight reference.
- void `set_weight_reference` (float value)

- Function to set the weight reference.*

 - void `reset_weight_reference` (void)

Function to reset the weight reference.
- bool `weight_reference_is_set` (void)

Function to send confirmation that the weight reference is set.
- void `setup_scale` (float calFactor)

Function to seting up the scale, is called in the begining of the program.
- bool `update_scale` (void)

Function for updating the scale value.
- float `get_weight` (void)

Function to get the scale reading.
- void `tare_scale` (void)

Function to tare the scale.
- bool `tare_complete` (void)

Function to signal that the scale has been tared.
- int `get_beer_cans_taken` (float referencWeight, float currentWeight)

Function to get the number of beer cans taken.

Variables

- HX711_ADC `scale`

5.13.1 Detailed Description

Author

Amal Araweelo Almis

5.13.2 Macro Definition Documentation

5.13.2.1 BEER_WEIGHT

```
#define BEER_WEIGHT 350
```

Define the weight of a beer.

5.13.2.2 HX711_DOUT

```
#define HX711_DOUT 4
```

5.13.2.3 HX711_SCK

```
#define HX711_SCK 5
```

5.13.2.4 SCALE_DEFAULT_SETTLE_TIME_MS

```
#define SCALE_DEFAULT_SETTLE_TIME_MS 3000
```

5.13.2.5 SCALE_TOL

```
#define SCALE_TOL 25
```

Define the uncertainty of the weight mesurment.

5.13.3 Function Documentation

5.13.3.1 get_beer_cans_taken()

```
int get_beer_cans_taken (
    float referencWeight,
    float currentWeight )
```

Function to get the number of beer cans taken.

Parameters

<i>referencWeight</i>	
<i>currentWeight</i>	

Returns

int

5.13.3.2 get_weight()

```
float get_weight (
    void )
```

Function to get the scale reading.

Returns

float

5.13.3.3 get_weight_reference()

```
float get_weight_reference (
    void )
```

Function to get the weight reference.

Returns

float

5.13.3.4 reset_weight_reference()

```
void reset_weight_reference (
    void )
```

Function to reset the weight reference.

5.13.3.5 set_weight_reference()

```
void set_weight_reference (
    float value )
```

Function to set the weight reference.

Parameters

<i>value</i>	
--------------	--

5.13.3.6 setup_scale()

```
void setup_scale (
    float calFactor )
```

Function to setting up the scale, is called in the beginning of the program.

Parameters

<i>calFactor</i>	
------------------	--

5.13.3.7 tare_complete()

```
bool tare_complete (
    void )
```

Function to signal that the scale has been tared.

Returns

true
false

5.13.3.8 tare_scale()

```
void tare_scale (
    void )
```

Function to tare the scale.

5.13.3.9 update_scale()

```
bool update_scale (
    void )
```

Function for updating the scale value.

Returns

true
false

5.13.3.10 `weight_reference_is_set()`

```
bool weight_reference_is_set (  
    void )
```

Function to send confirmation that the weight reference is set.

Returns

true
false

5.13.4 Variable Documentation

5.13.4.1 `scale`

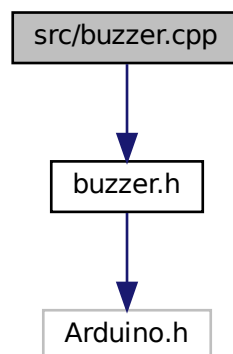
```
HX711_ADC scale [extern]
```

5.14 README.md File Reference

5.15 `src/buzzer.cpp` File Reference

```
#include "buzzer.h"
```

Include dependency graph for `buzzer.cpp`:



Functions

- void `play_warning` (unsigned long t)
Plays short sound when door is closed and about to be locked.
- void `play_unlock` ()
Plays an ascending tone, when the door unlocks.
- void `play_lock` ()
Play a descending tone, when the door locks.

Variables

- const int `BUZZERPIN` = 2
GPIO pin connected to the buzzer.
- const double `HIGH_TONE` = 1000
Frequency (Hz) used for high-tone buzzer sound.
- const double `LOW_TONE` = 600
Frequency (Hz) used for low-tone buzzer sound.
- const unsigned long `TONE_LENGTH` = 200
Duration of buzzer tone in milliseconds.

5.15.1 Function Documentation

5.15.1.1 `play_lock()`

```
void play_lock ( )
```

Play a descending tone, when the door locks.

5.15.1.2 `play_unlock()`

```
void play_unlock ( )
```

Plays an ascending tone, when the door unlocks.

5.15.1.3 `play_warning()`

```
void play_warning (
    unsigned long t )
```

Plays short sound when door is closed and about to be locked.

Parameters

<i>t</i>	
----------	--

5.15.2 Variable Documentation

5.15.2.1 BUZZERPIN

```
const int BUZZERPIN = 2
```

GPIO pin connected to the buzzer.

5.15.2.2 HIGH_TONE

```
const double HIGH_TONE = 1000
```

Frequency (Hz) used for high-tone buzzer sound.

5.15.2.3 LOW_TONE

```
const double LOW_TONE = 600
```

Frequency (Hz) used for low-tone buzzer sound.

5.15.2.4 TONE_LENGTH

```
const unsigned long TONE_LENGTH = 200
```

Duration of buzzer tone in milliseconds.

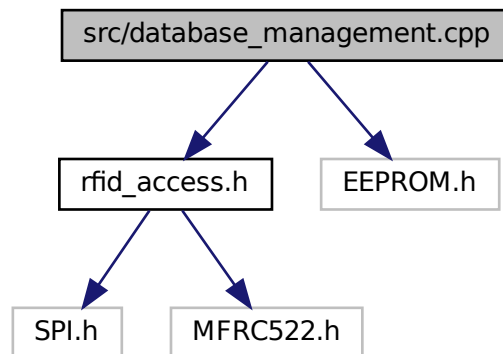
5.16 src/database_management.cpp File Reference

Used to read and write non-volatile memory on ESP8266.

```
#include "rfid_access.h"
```

```
#include <EEPROM.h>
```

Include dependency graph for database_management.cpp:



Functions

- void `user_management` (`RFIDCommand` incomingCommand, `User` *ptr, `MFRC522` &rfid)
Executes user-management actions based on the provided command.
- bool `remove_user` ()
Removes a user from the database.
- void `get_users_db` (`User` *ptr)
Copies the current user database to a provided buffer.
- void `print_all_users` (`User` *ptr)
Prints all users in the database to the serial interface.
- void `print_single_user` (`User` *ptr, int idx)
brief Prints a single user entry to the serial interface.
- void `print_uid` (byte *ptr)
Prints a UID buffer to the serial interface.
- int `read_integer` ()
Reads an integer from the serial interface.
- bool `read_confirmation` ()
Reads a confirmation input from the serial interface.
- int `find_empty_index` (`User` *ptr)
brief Finds an empty slot in the user database.
- int `count_rooms` (`User` *ptr)
brief Counts the number of occupied user entries in the database.

5.16.1 Detailed Description

Used to read and write non-volatile memory on ESP8266.

Authors

Anssi Sohlman,

5.16.2 Function Documentation

5.16.2.1 count_rooms()

```
int count_rooms (  
    User * ptr )
```

brief Counts the number of occupied user entries in the database.

Parameters

<i>ptr</i>	
------------	--

Returns

int

5.16.2.2 find_empty_index()

```
int find_empty_index (  
    User * ptr )
```

brief Finds an empty slot in the user database.

Parameters

<i>ptr</i>	
------------	--

Returns

int

5.16.2.3 get_users_db()

```
void get_users_db (
    User * ptr )
```

Copies the current user database to a provided buffer.

Parameters

<i>ptr</i>	
------------	--

5.16.2.4 print_all_users()

```
void print_all_users (
    User * ptr )
```

Prints all users in the database to the serial interface.

Parameters

<i>ptr</i>	
------------	--

5.16.2.5 print_single_user()

```
void print_single_user (
    User * ptr,
    int idx )
```

brief Prints a single user entry to the serial interface.

Parameters

<i>ptr</i>	
<i>idx</i>	

5.16.2.6 print_uid()

```
void print_uid (
    byte * ptr )
```

Prints a UID buffer to the serial interface.

Parameters

<i>ptr</i>	
------------	--

5.16.2.7 read_confirmation()

```
bool read_confirmation ( )
```

Reads a confirmation input from the serial interface.

Returns

true
false

5.16.2.8 read_integer()

```
int read_integer ( )
```

Reads an integer from the serial interface.

Returns

int

5.16.2.9 remove_user()

```
bool remove_user ( )
```

Removes a user from the database.

Returns

true
false

5.16.2.10 user_management()

```
void user_management (
    RFIDcommand cmd,
    User * ptr,
    MFRC522 & rfid )
```

Executes user-management actions based on the provided command.

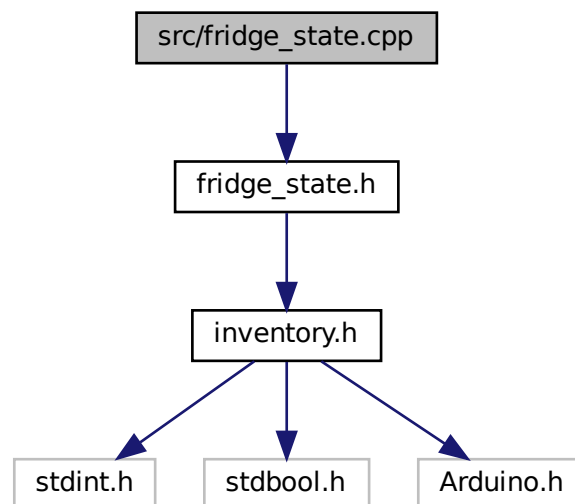
Parameters

<i>cmd</i>	
<i>ptr</i>	
<i>rfd</i>	

5.17 src/fridge_state.cpp File Reference

```
#include "fridge_state.h"
```

Include dependency graph for fridge_state.cpp:



Variables

- [inventory fridge](#)

Used to set up the fridge inventory.

5.17.1 Detailed Description

Author

Baldur G. Toftegaard

5.17.2 Variable Documentation

5.17.2.1 fridge

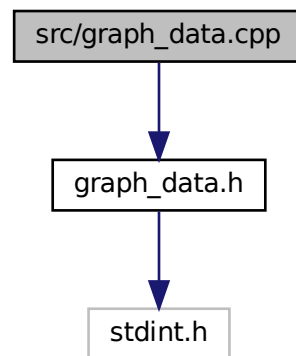
`inventory` fridge

Used to set up the fridge inventory.

5.18 src/graph_data.cpp File Reference

```
#include "graph_data.h"
```

Include dependency graph for graph_data.cpp:



Functions

- void `graph_add_to_room_green` (uint8_t roomNumber, int delta)
Adds a value to the green sales bar of a given room.
- void `graph_add_to_room_clasic` (uint8_t roomNumber, int delta)
Adds a value to the classic sales bar of a given room.

Variables

- int `greenHeight` [ROOM_COUNT]
Sales graph height data for green beer.
- int `classicHeight` [ROOM_COUNT]
Sales graph height data for classic beer.

5.18.1 Function Documentation

5.18.1.1 graph_add_to_room_clasic()

```
void graph_add_to_room_clasic (  
    uint8_t roomNumber,  
    int delta )
```

Adds a value to the classic sales bar of a given room.

Parameters

<i>roomNumber</i>	
<i>delta</i>	

5.18.1.2 graph_add_to_room_green()

```
void graph_add_to_room_green (
    uint8_t roomNumber,
    int delta )
```

Adds a value to the green sales bar of a given room.

Parameters

<i>roomNumber</i>	
<i>delta</i>	

5.18.2 Variable Documentation**5.18.2.1 classicHeight**

```
int classicHeight[ROOM_COUNT] [extern]
```

Sales graph height data for classic beer.

Height values for classic product sales per room.

Indexed by room number. Used by sale_html and the /saleHeights endpoint.

5.18.2.2 greenHeight

```
int greenHeight[ROOM_COUNT] [extern]
```

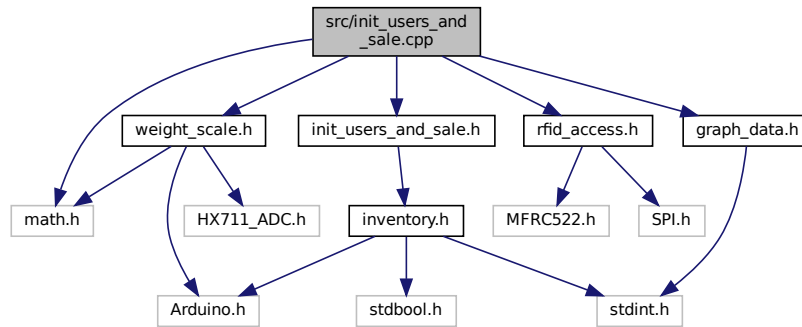
Sales graph height data for green beer.

Height values for green product sales per room.

Indexed by room number. Used by sale_html and the /saleHeights endpoint.

5.19 src/init_users_and_sale.cpp File Reference

```
#include "init_users_and_sale.h"
#include <math.h>
#include "weight_scale.h"
#include "rfid_access.h"
#include "graph_data.h"
Include dependency graph for init_users_and_sale.cpp:
```



Functions

- void [init_users_and_products](#) ()
Not used in prototype, should be moved to [inventory.cpp](#).
- static float [read_current_weight_blocking](#) (uint32_t timeoutMs=1200)
- void [perform_sale](#) ([inventory](#) *fridge_inventory)
Performs a sale between a user and the fridge.

5.19.1 Function Documentation

5.19.1.1 [init_users_and_products](#)()

```
void init_users_and_products ( )
```

Not used in prototype, should be moved to [inventory.cpp](#).

Old function for initializing users and products.

5.19.1.2 [perform_sale](#)()

```
void perform_sale (
    inventory * fridge_inventory )
```

Performs a sale between a user and the fridge.

Parameters

<i>weight</i>	
<i>user_id</i>	
<i>fridge_inventory</i>	

Parameters

<i>fridge_inventory</i>	Inventory that the sale shuld remove item from
-------------------------	--

5.19.1.3 read_current_weight_blocking()

```
static float read_current_weight_blocking (
    uint32_t timeoutMs = 1200 ) [static]
```

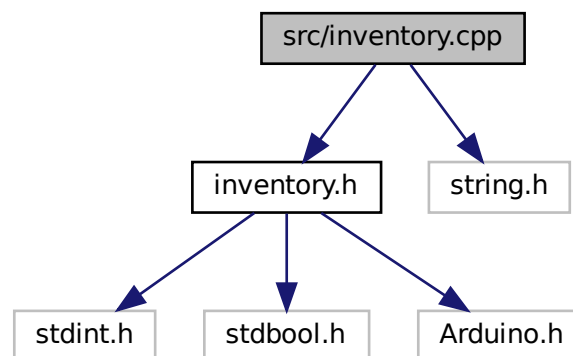
5.20 src/inventory.cpp File Reference

Functions responsible for keeping track of the fridge inventory.

```
#include "inventory.h"
```

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

Include dependency graph for inventory.cpp:



Functions

- void `inventory_init` (`inventory *inventory`)
Initializes an inventory structure.
- `product inventory_make_product` (`const char *product_name`, `beverage_type` type, `uint16_t` weight, `uint8_t` price)
Creates a new product instance.
- bool `inventory_add_product` (`inventory *inventory`, `product product`, `uint16_t` quantity)
Adds a new product to the inventory.
- bool `inventory_remove_product` (`inventory *inventory`, `product beverage`)
Removes a product entirely from the inventory.
- bool `inventory_add_beverage` (`inventory *inventory`, `product beverag`, `uint8_t` amount)
- bool `inventory_remove_beverage` (`inventory *inventory`, `product beverag`, `uint8_t` amount)
Function for removing from the amount of a beverage in an inventory.
- void `inventory_print` (`inventory *inventory`)
Function to print a users inventory.

5.20.1 Detailed Description

Functions responsible for keeping track of the fridge inventory.

Author

Baldur G. Toftegaard

5.20.2 Function Documentation

5.20.2.1 `inventory_add_beverage()`

```
bool inventory_add_beverage (  
    inventory * inventory,  
    product beverag,  
    uint8_t amount )
```

5.20.2.2 `inventory_add_product()`

```
bool inventory_add_product (  
    inventory * inventory,  
    product product,  
    uint16_t quantity )
```

Adds a new product to the inventory.

Parameters

<i>inventory</i>	Inventory to add the product to.
<i>product</i>	Product to add.
<i>quantity</i>	Initial quantity of the product.

Returns

true Product successfully added.
false Error occurred (inventory full or duplicate product).

5.20.2.3 inventory_init()

```
void inventory_init (
    inventory * inventory )
```

Initializes an inventory structure.

Clears internal state and prepares the inventory for use.

Parameters

<i>inventory</i>	Pointer to inventory instance to initialize.
------------------	--

5.20.2.4 inventory_make_product()

```
product inventory_make_product (
    const char * name,
    beverage_type type,
    uint16_t weight,
    uint8_t price )
```

Creates a new product instance.

Parameters

<i>name</i>	Display name of the product.
<i>type</i>	Beverage type.
<i>weight</i>	Weight of the product.
<i>price</i>	Price of the product.

Returns

Initialized product instance.

5.20.2.5 inventory_print()

```
void inventory_print (
    inventory * inventory )
```

Function to print a users inventory.

Parameters

<i>inventory</i>	Inventory to print.
------------------	---------------------

5.20.2.6 inventory_remove_beverage()

```
bool inventory_remove_beverage (
    inventory * inventory,
    product beverage,
    uint8_t amount )
```

Function for removing from the amount of a beverage in an inventory.

Parameters

<i>inventory</i>	Inventory to modify.
<i>beverage</i>	Beverage to update.
<i>amount</i>	Quantity to remove.

Returns

true Quantity updated successfully.
false Product not found or insufficient stock.

5.20.2.7 inventory_remove_product()

```
bool inventory_remove_product (
    inventory * inventory,
    product beverage )
```

Removes a product entirely from the inventory.

Parameters

<i>inventory</i>	Inventory to remove the product from.
<i>beverage</i>	Product to remove.

Returns

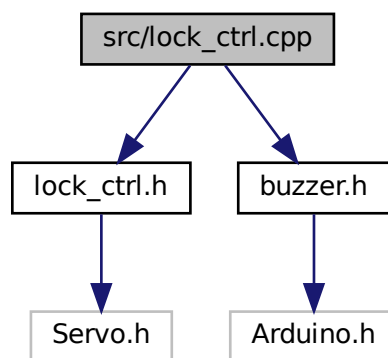
true Product removed successfully.
false Product not found.

5.21 src/lock_ctrl.cpp File Reference

```
#include "lock_ctrl.h"
```

```
#include "buzzer.h"
```

Include dependency graph for lock_ctrl.cpp:



Functions

- void `lock_ctrl_init` ()
Initializes the lock control module.
- void `lock_door` ()
Locks the door using the servo.
- void `unlock_door` ()
Unlocks the door using the servo.
- bool `is_box_closed` ()
Checks whether the box is closed.
- void `play_open` ()
Plays the sound effect for door opening.
- void `play_close` ()
Play sound effect when the door closes.

Variables

- static Servo `lockServo`
- static const int `UNLOCK_POS` = 0
Servo position corresponding to the unlocked state (mechanically fixed).
- static const int `LOCK_POS` = 100
Servo position corresponding to the locked state.
- const int `BUZZER` = 2
GPIO pin connected to the buzzer.
- const double `HIGH_TONE` = 1000
Frequency (Hz) used for the high-tone buzzer sound.
- const double `LOW_TONE` = 600
Frequency (Hz) used for the low-tone buzzer sound.
- const unsigned long `TONE_LENGTH` = 200
Duration of buzzer tone in milliseconds.
- static bool `boxClosed` = false
Internal latched state indicating whether the box is closed.

5.21.1 Detailed Description

Authors

Amal Araweelo Almis

5.21.2 Function Documentation

5.21.2.1 `is_box_closed()`

```
bool is_box_closed ( )
```

Checks whether the box is closed.

Uses the light sensor to determine door state.

Returns

true Box is closed.

false Box is open.

5.21.2.2 `lock_ctrl_init()`

```
void lock_ctrl_init ( )
```

Initializes the lock control module.

Must be called once during system startup before any other lock control functions are used.

5.21.2.3 lock_door()

```
void lock_door ( )
```

Locks the door using the servo.

5.21.2.4 play_close()

```
void play_close ( )
```

Play sound effect when the door closes.

5.21.2.5 play_open()

```
void play_open ( )
```

Plays the sound effect for door opening.

5.21.2.6 unlock_door()

```
void unlock_door ( )
```

Unlocks the door using the servo.

5.21.3 Variable Documentation

5.21.3.1 boxClosed

```
bool boxClosed = false [static]
```

Internal latched state indicating whether the box is closed.

5.21.3.2 BUZZER

```
const int BUZZER = 2
```

GPIO pin connected to the buzzer.

Note

This pin choice may cause reset issues on some boards.

5.21.3.3 HIGH_TONE

```
const double HIGH_TONE = 1000
```

Frequency (Hz) used for the high-tone buzzer sound.

5.21.3.4 LOCK_POS

```
const int LOCK_POS = 100 [static]
```

Servo position corresponding to the locked state.

5.21.3.5 lockServo

```
Servo lockServo [static]
```

5.21.3.6 LOW_TONE

```
const double LOW_TONE = 600
```

Frequency (Hz) used for the low-tone buzzer sound.

5.21.3.7 TONE_LENGTH

```
const unsigned long TONE_LENGTH = 200
```

Duration of buzzer tone in milliseconds.

5.21.3.8 UNLOCK_POS

```
const int UNLOCK_POS = 0 [static]
```

Servo position corresponding to the unlocked state (mechanically fixed).

5.22 src/main.cpp File Reference

Combined: Web server + Graph + Scale + RFID access + Lock control.

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
#include <ESP8266WiFiMulti.h>
#include "index_html.h"
#include "sale_html.h"
#include "STYLE_CSS.h"
#include "LOGIN_HTML.h"
#include "ADMIN_HTML.h"
#include "graph_data.h"
#include "inventory.h"
#include "init_users_and_sale.h"
#include "weight_scale.h"
#include "fridge_state.h"
#include "rfid_access.h"
#include "lock_ctrl.h"
#include "buzzer.h"
```

Include dependency graph for main.cpp:



Functions

- void `print_graph_arrays` ()
Prints the current graph height arrays.
- ESP8266WebServer `server` (80)
- MFRC522 `rfid` (SS_PIN, RST_PIN)
- static void `connect_wifi_and_start_mdns` ()
- static void `setup_web_routes` ()
- static void `setup_inventory_and_scale` ()
- static void `setup_rfid_and_lock` ()
- void `setup` ()
- void `loop` ()

Variables

- static const float `CAL_FACTOR` = 22.9f
Calibration factor used for weight or sensor calculations.
- static const uint16_t `START_BEER_QTY` = 20
Initial quantity of beer available at system startup.
- const char * `WIFI_SSID` = "Baldur's A56"
Wi-Fi network SSID used by the device.
- const char * `WIFI_PASS` = "MyPasskeyA56"
Wi-Fi network password used by the device.
- int `greenHeight` [`ROOM_COUNT`]
Sales graph height data for green beer.
- int `classicHeight` [`ROOM_COUNT`]
Sales graph height data for classic beer.
- `product demo_beer`
- `RFIDcommand activeCommand` = `CMD_NONE`
- bool `doorUnlocked` = false
- unsigned long `doorCloseTimer` = 0

5.22.1 Detailed Description

Combined: Web server + Graph + Scale + RFID access + Lock control.

Author

Baldur G. Toftegaard

5.22.2 Function Documentation

5.22.2.1 `connect_wifi_and_start_mdns()`

```
static void connect_wifi_and_start_mdns ( ) [static]
```

5.22.2.2 `loop()`

```
void loop ( )
```

5.22.2.3 `print_graph_arrays()`

```
void print_graph_arrays ( )
```

Prints the current graph height arrays.

5.22.2.4 rfid()

```
MFRC522 rfid (
    SS_PIN ,
    RST_PIN )
```

5.22.2.5 server()

```
ESP8266WebServer server (
    80 )
```

5.22.2.6 setup()

```
void setup ( )
```

5.22.2.7 setup_inventory_and_scale()

```
static void setup_inventory_and_scale ( ) [static]
```

5.22.2.8 setup_rfid_and_lock()

```
static void setup_rfid_and_lock ( ) [static]
```

5.22.2.9 setup_web_routes()

```
static void setup_web_routes ( ) [static]
```

5.22.3 Variable Documentation

5.22.3.1 activeCommand

```
RFIDcommand activeCommand = CMD_NONE
```

5.22.3.2 CAL_FACTOR

```
const float CAL_FACTOR = 22.9f [static]
```

Calibration factor used for weight or sensor calculations.

5.22.3.3 classicHeight

```
int classicHeight[ROOM_COUNT]
```

Initial value:

```
= {  
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1  
}
```

Sales graph height data for classic beer.

Height values for classic product sales per room.

Indexed by room number. Used by sale_html and the /saleHeights endpoint.

5.22.3.4 demo_beer

```
product demo_beer
```

5.22.3.5 doorCloseTimer

```
unsigned long doorCloseTimer = 0
```

5.22.3.6 doorUnlocked

```
bool doorUnlocked = false
```

5.22.3.7 greenHeight

```
int greenHeight[ROOM_COUNT]
```

Initial value:

```
= {  
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1  
}
```

Sales graph height data for green beer.

Height values for green product sales per room.

Indexed by room number. Used by sale_html and the /saleHeights endpoint.

5.22.3.8 START_BEER_QTY

```
const uint16_t START_BEER_QTY = 20 [static]
```

Initial quantity of beer available at system startup.

5.22.3.9 WIFI_PASS

```
const char* WIFI_PASS = "MyPasskeyA56"
```

Wi-Fi network password used by the device.

5.22.3.10 WIFI_SSID

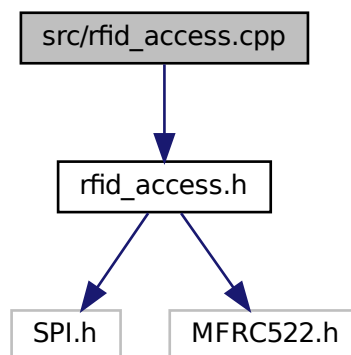
```
const char* WIFI_SSID = "Baldur's A56"
```

Wi-Fi network SSID used by the device.

5.23 src/rfid_access.cpp File Reference

```
#include "rfid_access.h"
```

Include dependency graph for rfid_access.cpp:



Functions

- `RFIDcommand check_command ()`
brief Reads a command from the serial interface and maps it to an RFIDcommand.
- `void setup_RFID_reader (MFRC522 &rfid)`
Initializes SPI and the MFRC522 RFID reader.
- `bool add_user (MFRC522 &rfid)`
Adds a new user by reading room number and scanning an RFID tag.
- `bool validate_rfid (MFRC522 myRFID)`
Validates an RFID tag against the registered user database.
- `bool compare_UID (byte *uid1, byte *uid2)`
Compares two RFID UUIDs.
- `bool read_RFID_tag (MFRC522 &rfid, byte *uidBuffer)`
Reads an RFID tag UUID from the MFRC522 reader.
- `void display_commands ()`
Prints the available serial commands.
- `void display_commands_um ()`
Prints the available serial commands for user-management mode.
- `void rfid_set_last_uid (const byte *uidIn)`
Function for storing the last used RFID.
- `bool rfid_get_last_uid (byte *uidOut)`
Function for restoring the last used RFID.

Variables

- `User users [MAX_ROOMS]`
Global user database array.
- `int userCount = 0`
Number of currently registered users.
- `static byte lastUID [UUID_LENGTH]`
- `static bool hasUID = false`

5.23.1 Detailed Description

Author

Amal Araweelo Almis

Baldur G. Toftegaard

5.23.2 Function Documentation

5.23.2.1 add_user()

```
bool add_user (
    MFRC522 & rfid )
```

Adds a new user by reading room number and scanning an RFID tag.

Parameters

<i>rfid</i>	
-------------	--

Returns

true
false

5.23.2.2 check_command()

```
RFIDcommand check_command (  
    void )
```

brief Reads a command from the serial interface and maps it to an RFIDcommand.

Returns

RFIDcommand

5.23.2.3 compare_UID()

```
bool compare_UID (  
    byte * uid1,  
    byte * uid2 )
```

Compares two RFID UUIDs.

Parameters

<i>uid1</i>	
<i>uid2</i>	

Returns

true
false

5.23.2.4 display_commands()

```
void display_commands (  
    void )
```

Prints the available serial commands.

5.23.2.5 display_commands_um()

```
void display_commands_um ( )
```

Prints the available serial commands for user-management mode.

5.23.2.6 read_RFID_tag()

```
bool read_RFID_tag (
    MFRC522 & rfid,
    byte * uidBuffer )
```

Reads an RFID tag UID from the MFRC522 reader.

Parameters

<i>rfid</i>	
<i>uidBuffer</i>	

Returns

true
false

5.23.2.7 rfid_get_last_uid()

```
bool rfid_get_last_uid (
    byte * uidOut )
```

Function for restoring the last used RFID.

Parameters

<i>uidOut</i>	
---------------	--

Returns

true
false

5.23.2.8 rfid_set_last_uid()

```
void rfid_set_last_uid (
    const byte * uidIn )
```

Function for storing the last used RFID.

Parameters

<i>uidOut</i>	
---------------	--

Returns

true

false

5.23.2.9 setup_RFID_reader()

```
void setup_RFID_reader (
    MFRC522 & rfid )
```

Initializes SPI and the MFRC522 RFID reader.

Parameters

<i>rfid</i>	
-------------	--

5.23.2.10 validate_rfid()

```
bool validate_rfid (
    MFRC522 myRFID )
```

Validates an RFID tag against the registered user database.

Parameters

<i>myRFID</i>	
---------------	--

Returns

true

false

5.23.3 Variable Documentation

5.23.3.1 hasUID

```
bool hasUID = false [static]
```

5.23.3.2 lastUID

```
byte lastUID[UID_LENGTH] [static]
```

5.23.3.3 userCount

```
int userCount = 0
```

Number of currently registered users.

5.23.3.4 users

```
User users[MAX_ROOMS]
```

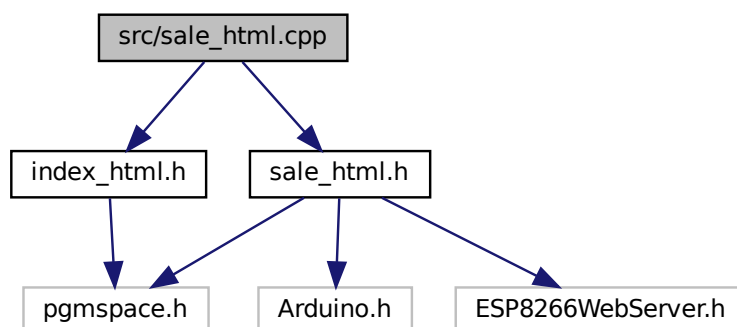
Global user database array.

5.24 src/sale_html.cpp File Reference

```
#include "index_html.h"
```

```
#include "sale_html.h"
```

Include dependency graph for sale_html.cpp:



Functions

- void `send_sale_html_graph` (ESP8266WebServer &`server`, uint8_t `room_number`, const char *`bar_type`, int `bar_height`)
Sends a single sales bar element to the client.
- void `send_sale_html_page` (ESP8266WebServer &`server`, uint8_t `room_count`, const int *`greenHeight`, const int *`classicHeights`)
Sends the complete sales graph page.

Variables

- const char SALE_BOX_START[] `PROGMEM` = R"rawliteral(<div class="sale_box">)rawliteral"
Opening container for the sales graph.

5.24.1 Detailed Description

Author

Baldur G. Toftegaard

5.24.2 Function Documentation

5.24.2.1 send_sale_html_graph()

```
void send_sale_html_graph (
    ESP8266WebServer & server,
    uint8_t room_number,
    const char * bar_type,
    int bar_height )
```

Sends a single sales bar element to the client.

Parameters

<code>server</code>	
<code>room_number</code>	
<code>bar_type</code>	
<code>bar_height</code>	

Parameters

<code>server</code>	The server
<code>room_number</code>	Number of the relevant room
<code>bar_type</code>	The type of the bar graph
<code>bar_height</code>	The height of the bar graph

5.24.2.2 send_sale_html_page()

```
void send_sale_html_page (
    ESP8266WebServer & server,
    uint8_t room_count,
    const int * greenHeight,
    const int * classicHeights )
```

Sends the complete sales graph page.

Parameters

<i>server</i>	
<i>room_count</i>	
<i>greenHeight</i>	
<i>classicHeights</i>	

Parameters

<i>server</i>	The server
<i>room_count</i>	The number of rooms
<i>greenHeight</i>	The hight of the green bar
<i>classicHeights</i>	The hight of the clasic bar (not used in prototype)

5.24.3 Variable Documentation

5.24.3.1 PROGMEM

```
const char SALE_BOX_STOP [ ] PROGMEM = R"rawliteral( <div class="sale_box">)rawliteral"
```

Opening container for the sales graph.

Closing container for the complete sales graph.

Closing container for a single room graph.

Closing fragment for a single sales bar.

HTML fragment defining the height style of a bar.

HTML fragment defining the CSS class type for a bar.

HTML fragment for the room identifier.

Opening container for a single room graph.

Opening container for the sales graph.

Opening container for the sales graph.

String used for the HTML footer.

This is responsible for updating the graphs.

Opening container for the sales graph.

Interpreted as a string by the compiler.

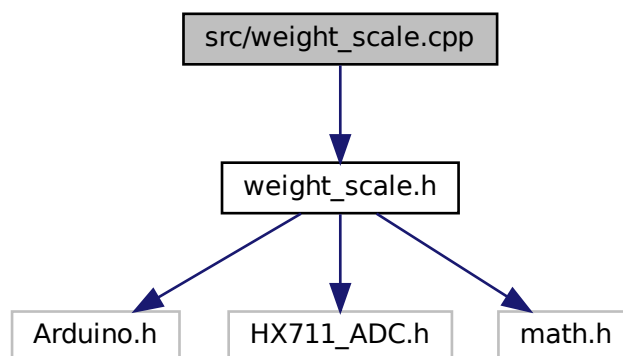
Opening container for the sales graph.

is handled like a string by the compiler

5.25 src/weight_scale.cpp File Reference

```
#include "weight_scale.h"
```

Include dependency graph for weight_scale.cpp:



Functions

- HX711_ADC [scale](#) (HX711_DOUT, HX711_SCK)
- float [get_weight_reference](#) (void)
Function to get the weight reference.
- void [set_weight_reference](#) (float value)
Function to set the weight reference.
- void [reset_weight_reference](#) (void)
Function to reset the weight reference.
- bool [weight_reference_is_set](#) (void)
Function to send confirmation that the weight reference is set.
- void [setup_scale](#) (float calFactor)

Function to seting up the scale, is called in the begining of the program.

- bool `update_scale` ()

Function for updating the scale value.

- float `get_weight` ()

Function to get the scale reading.

- void `tare_scale` ()

Function to tare the scale.

- bool `tare_complete` ()

Function to signal that the scale has been tared.

- int `get_beer_cans_taken` (float `referenceWeight`, float `currentWeight`)

Function to get the number of beer cans taken.

Variables

- static float `g_referenceWeight` = NAN

5.25.1 Detailed Description

Author

Amal Araweelo Almis

5.25.2 Function Documentation

5.25.2.1 `get_beer_cans_taken()`

```
int get_beer_cans_taken (
    float referencWeight,
    float currentWeight )
```

Function to get the number of beer cans taken.

Parameters

<i>referencWeight</i>	
<i>currentWeight</i>	

Returns

int

5.25.2.2 get_weight()

```
float get_weight (
    void )
```

Function to get the scale reading.

Returns

float

5.25.2.3 get_weight_reference()

```
float get_weight_reference (
    void )
```

Function to get the weight reference.

Returns

float

5.25.2.4 reset_weight_reference()

```
void reset_weight_reference (
    void )
```

Function to reset the weight reference.

5.25.2.5 scale()

```
HX711_ADC scale (
    HX711_DOUT ,
    HX711_SCK )
```

5.25.2.6 set_weight_reference()

```
void set_weight_reference (
    float value )
```

Function to set the weight reference.

Parameters

<i>value</i>	
--------------	--

5.25.2.7 setup_scale()

```
void setup_scale (
    float calFactor )
```

Function to setting up the scale, is called in the begining of the program.

Parameters

<i>calFactor</i>	
------------------	--

5.25.2.8 tare_complete()

```
bool tare_complete (
    void )
```

Function to signal that the scale has been tared.

Returns

true
false

5.25.2.9 tare_scale()

```
void tare_scale (
    void )
```

Function to tare the scale.

5.25.2.10 update_scale()

```
bool update_scale (
    void )
```

Function for updating the scale value.

Returns

true
false

5.25.2.11 weight_reference_is_set()

```
bool weight_reference_is_set (  
    void )
```

Function to send confirmation that the weight reference is set.

Returns

true
false

5.25.3 Variable Documentation

5.25.3.1 g_referenceWeight

```
float g_referenceWeight = NAN [static]
```


Index

- activeCommand
 - main.cpp, [75](#)
- add_user
 - rfid_access.cpp, [78](#)
 - rfid_access.h, [37](#)
- admin_html.h
 - PROGMEM, [14](#)
- balance
 - User, [12](#)
- beer
 - inventory.h, [26](#)
- BEER_WEIGHT
 - weight_scale.h, [50](#)
- beverage
 - products_stocked, [11](#)
- beverage_type
 - inventory.h, [26](#)
- beverage_variant
 - product, [9](#)
- boxClosed
 - lock_ctrl.cpp, [71](#)
- BUZZER
 - lock_ctrl.cpp, [71](#)
- buzzer.cpp
 - BUZZERPIN, [56](#)
 - HIGH_TONE, [56](#)
 - LOW_TONE, [56](#)
 - play_lock, [55](#)
 - play_unlock, [55](#)
 - play_warning, [55](#)
 - TONE_LENGTH, [56](#)
- buzzer.h
 - BUZZER_H, [15](#)
 - play_lock, [15](#)
 - play_unlock, [16](#)
 - play_warning, [16](#)
- BUZZER_H
 - buzzer.h, [15](#)
- BUZZERPIN
 - buzzer.cpp, [56](#)
- CAL_FACTOR
 - main.cpp, [75](#)
- check_command
 - rfid_access.cpp, [79](#)
 - rfid_access.h, [38](#)
- cider
 - inventory.h, [26](#)
- classicHeight
 - graph_data.cpp, [63](#)
 - graph_data.h, [20](#)
 - main.cpp, [76](#)
- CLOSED_THRESHOLD
 - lock_ctrl.h, [31](#)
- CMD_ADD_USER
 - rfid_access.h, [37](#)
- CMD_CONFIRM
 - rfid_access.h, [37](#)
- CMD_LOCK
 - rfid_access.h, [37](#)
- CMD_NONE
 - rfid_access.h, [37](#)
- CMD_OPEN
 - rfid_access.h, [37](#)
- CMD_PRINT
 - rfid_access.h, [37](#)
- CMD_REMOVE_USER
 - rfid_access.h, [37](#)
- compare_UID
 - rfid_access.cpp, [79](#)
 - rfid_access.h, [38](#)
- connect_wifi_and_start_mdns
 - main.cpp, [74](#)
- count_rooms
 - database_management.cpp, [58](#)
 - rfid_access.h, [38](#)
- current_quantity
 - products_stocked, [11](#)
- database_management.cpp
 - count_rooms, [58](#)
 - find_empty_index, [58](#)
 - get_users_db, [58](#)
 - print_all_users, [59](#)
 - print_single_user, [59](#)
 - print_uid, [59](#)
 - read_confirmation, [60](#)
 - read_integer, [60](#)
 - remove_user, [60](#)
 - user_management, [60](#)
- demo_beer
 - main.cpp, [76](#)
- display_commands
 - rfid_access.cpp, [79](#)
 - rfid_access.h, [39](#)
- display_commands_um
 - rfid_access.cpp, [80](#)
 - rfid_access.h, [39](#)
- doorCloseTimer

- main.cpp, 76
- doorUnlocked
 - main.cpp, 76
- find_empty_index
 - database_management.cpp, 58
 - rfid_access.h, 39
- fridge
 - fridge_state.cpp, 61
 - fridge_state.h, 18
- fridge_state.cpp
 - fridge, 61
- fridge_state.h
 - fridge, 18
- g_referenceWeight
 - weight_scale.cpp, 89
- get_beer_cans_taken
 - weight_scale.cpp, 86
 - weight_scale.h, 51
- get_users_db
 - database_management.cpp, 58
 - rfid_access.h, 39
- get_weight
 - weight_scale.cpp, 86
 - weight_scale.h, 51
- get_weight_reference
 - weight_scale.cpp, 87
 - weight_scale.h, 52
- graph_add_to_room_clasic
 - graph_data.cpp, 62
 - graph_data.h, 19
- graph_add_to_room_green
 - graph_data.cpp, 63
 - graph_data.h, 19
- graph_data.cpp
 - classicHeight, 63
 - graph_add_to_room_clasic, 62
 - graph_add_to_room_green, 63
 - greenHeight, 63
- graph_data.h
 - classicHeight, 20
 - graph_add_to_room_clasic, 19
 - graph_add_to_room_green, 19
 - greenHeight, 20
 - print_graph_arrays, 20
 - ROOM_COUNT, 19
- greenHeight
 - graph_data.cpp, 63
 - graph_data.h, 20
 - main.cpp, 76
- hasUID
 - rfid_access.cpp, 82
- HIGH_TONE
 - buzzer.cpp, 56
 - lock_ctrl.cpp, 72
- HX711_DOUT
 - weight_scale.h, 50
- HX711_SCK
 - weight_scale.h, 50
- include/admin_html.h, 13
- include/buzzer.h, 14
- include/fridge_state.h, 16
- include/graph_data.h, 18
- include/index_html.h, 21
- include/init_users_and_sale.h, 22
- include/inventory.h, 24
- include/lock_ctrl.h, 29
- include/login_html.h, 33
- include/rfid_access.h, 34
- include/sale_html.h, 45
- include/style_css.h, 48
- include/weight_scale.h, 49
- index_html.h
 - PROGMEM, 22
- init_users_and_products
 - init_users_and_sale.cpp, 64
 - init_users_and_sale.h, 23
- init_users_and_sale.cpp
 - init_users_and_products, 64
 - perform_sale, 64
 - read_current_weight_blocking, 65
- init_users_and_sale.h
 - init_users_and_products, 23
 - number_of_users, 23
 - perform_sale, 24
- inventory, 7
 - number_of_products_stocked, 8
 - products_in_inventory, 8
 - room_number, 8
- inventory.cpp
 - inventory_add_beverage, 66
 - inventory_add_product, 66
 - inventory_init, 67
 - inventory_make_product, 67
 - inventory_print, 68
 - inventory_remove_beverage, 68
 - inventory_remove_product, 68
- inventory.h
 - beer, 26
 - beverage_type, 26
 - cider, 26
 - inventory_add_beverage, 26
 - inventory_add_product, 27
 - INVENTORY_CAPACITY, 26
 - inventory_init, 27
 - inventory_make_product, 28
 - inventory_print, 28
 - inventory_remove_beverage, 28
 - inventory_remove_product, 29
 - limfjords_porter, 26
 - other, 26
 - soda, 26
- inventory_add_beverage
 - inventory.cpp, 66
 - inventory.h, 26

- inventory_add_product
 - inventory.cpp, 66
 - inventory.h, 27
- INVENTORY_CAPACITY
 - inventory.h, 26
- inventory_init
 - inventory.cpp, 67
 - inventory.h, 27
- inventory_make_product
 - inventory.cpp, 67
 - inventory.h, 28
- inventory_print
 - inventory.cpp, 68
 - inventory.h, 28
- inventory_remove_beverage
 - inventory.cpp, 68
 - inventory.h, 28
- inventory_remove_product
 - inventory.cpp, 68
 - inventory.h, 29
- is_box_closed
 - lock_ctrl.cpp, 70
 - lock_ctrl.h, 31
- lastUID
 - rfid_access.cpp, 82
- limfjords_porter
 - inventory.h, 26
- lock_ctrl.cpp
 - boxClosed, 71
 - BUZZER, 71
 - HIGH_TONE, 72
 - is_box_closed, 70
 - lock_ctrl_init, 70
 - lock_door, 70
 - LOCK_POS, 72
 - lockServo, 72
 - LOW_TONE, 72
 - play_close, 71
 - play_open, 71
 - TONE_LENGTH, 72
 - unlock_door, 71
 - UNLOCK_POS, 72
- lock_ctrl.h
 - CLOSED_THRESHOLD, 31
 - is_box_closed, 31
 - lock_ctrl_init, 32
 - lock_door, 32
 - OPEN_THRESHOLD, 31
 - play_close, 32
 - play_open, 32
 - SERVO_PIN, 31
 - unlock_door, 32
- lock_ctrl_init
 - lock_ctrl.cpp, 70
 - lock_ctrl.h, 32
- lock_door
 - lock_ctrl.cpp, 70
 - lock_ctrl.h, 32
- LOCK_POS
 - lock_ctrl.cpp, 72
- lockServo
 - lock_ctrl.cpp, 72
- login_html.h
 - PROGMEM, 33
- loop
 - main.cpp, 74
- LOW_TONE
 - buzzer.cpp, 56
 - lock_ctrl.cpp, 72
- main.cpp
 - activeCommand, 75
 - CAL_FACTOR, 75
 - classicHeight, 76
 - connect_wifi_and_start_mdns, 74
 - demo_beer, 76
 - doorCloseTimer, 76
 - doorUnlocked, 76
 - greenHeight, 76
 - loop, 74
 - print_graph_arrays, 74
 - rfid, 74
 - server, 75
 - setup, 75
 - setup_inventory_and_scale, 75
 - setup_rfid_and_lock, 75
 - setup_web_routes, 75
 - START_BEER_QTY, 76
 - WIFI_PASS, 77
 - WIFI_SSID, 77
- MAX_ROOMS
 - rfid_access.h, 36
- name
 - product, 9
- number_of_products_stocked
 - inventory, 8
- number_of_users
 - init_users_and_sale.h, 23
- OPEN_THRESHOLD
 - lock_ctrl.h, 31
- original_quantity
 - products_stocked, 11
- other
 - inventory.h, 26
- perform_sale
 - init_users_and_sale.cpp, 64
 - init_users_and_sale.h, 24
- play_close
 - lock_ctrl.cpp, 71
 - lock_ctrl.h, 32
- play_lock
 - buzzer.cpp, 55
 - buzzer.h, 15
- play_open

- lock_ctrl.cpp, 71
- lock_ctrl.h, 32
- play_unlock
 - buzzer.cpp, 55
 - buzzer.h, 16
- play_warning
 - buzzer.cpp, 55
 - buzzer.h, 16
- price
 - product, 9
- print_all_users
 - database_management.cpp, 59
 - rfid_access.h, 41
- print_graph_arrays
 - graph_data.h, 20
 - main.cpp, 74
- print_single_user
 - database_management.cpp, 59
 - rfid_access.h, 41
- print_uid
 - database_management.cpp, 59
 - rfid_access.h, 41
- product, 8
 - beverage_variant, 9
 - name, 9
 - price, 9
 - weight, 9
- products_in_inventory
 - inventory, 8
- products_stocked, 10
 - beverage, 11
 - current_quantity, 11
 - original_quantity, 11
- PROGMEM
 - admin_html.h, 14
 - index_html.h, 22
 - login_html.h, 33
 - sale_html.cpp, 84
 - sale_html.h, 47
 - style_css.h, 48
- read_confirmation
 - database_management.cpp, 60
 - rfid_access.h, 41
- read_current_weight_blocking
 - init_users_and_sale.cpp, 65
- read_integer
 - database_management.cpp, 60
 - rfid_access.h, 42
- read_RFID_tag
 - rfid_access.cpp, 80
 - rfid_access.h, 42
- README.md, 54
- remove_user
 - database_management.cpp, 60
 - rfid_access.h, 42
- reset_weight_reference
 - weight_scale.cpp, 87
 - weight_scale.h, 52
- rfid
 - main.cpp, 74
- rfid_access.cpp
 - add_user, 78
 - check_command, 79
 - compare_UID, 79
 - display_commands, 79
 - display_commands_um, 80
 - hasUID, 82
 - lastUID, 82
 - read_RFID_tag, 80
 - rfid_get_last_uid, 80
 - rfid_set_last_uid, 80
 - setup_RFID_reader, 81
 - userCount, 82
 - users, 82
 - validate_rfid, 81
- rfid_access.h
 - add_user, 37
 - check_command, 38
 - CMD_ADD_USER, 37
 - CMD_CONFIRM, 37
 - CMD_LOCK, 37
 - CMD_NONE, 37
 - CMD_OPEN, 37
 - CMD_PRINT, 37
 - CMD_REMOVE_USER, 37
 - compare_UID, 38
 - count_rooms, 38
 - display_commands, 39
 - display_commands_um, 39
 - find_empty_index, 39
 - get_users_db, 39
 - MAX_ROOMS, 36
 - print_all_users, 41
 - print_single_user, 41
 - print_uid, 41
 - read_confirmation, 41
 - read_integer, 42
 - read_RFID_tag, 42
 - remove_user, 42
 - rfid_get_last_uid, 43
 - rfid_set_last_uid, 43
 - RFIDcommand, 37
 - RST_PIN, 36
 - setup_RFID_reader, 43
 - SS_PIN, 36
 - UID_LENGTH, 37
 - user_management, 44
 - userCount, 44
 - users, 44
 - validate_rfid, 44
- rfid_get_last_uid
 - rfid_access.cpp, 80
 - rfid_access.h, 43
- rfid_set_last_uid
 - rfid_access.cpp, 80
 - rfid_access.h, 43

- RFIDcommand
 - rfid_access.h, 37
- ROOM_COUNT
 - graph_data.h, 19
- room_number
 - inventory, 8
- roomNumber
 - User, 12
- RST_PIN
 - rfid_access.h, 36
- sale_html.cpp
 - PROGMEM, 84
 - send_sale_html_graph, 83
 - send_sale_html_page, 84
- sale_html.h
 - PROGMEM, 47
 - send_sale_html_graph, 46
 - send_sale_html_page, 46
- scale
 - weight_scale.cpp, 87
 - weight_scale.h, 54
- SCALE_DEFAULT_SETTLE_TIME_MS
 - weight_scale.h, 51
- SCALE_TOL
 - weight_scale.h, 51
- send_sale_html_graph
 - sale_html.cpp, 83
 - sale_html.h, 46
- send_sale_html_page
 - sale_html.cpp, 84
 - sale_html.h, 46
- server
 - main.cpp, 75
- SERVO_PIN
 - lock_ctrl.h, 31
- set_weight_reference
 - weight_scale.cpp, 87
 - weight_scale.h, 52
- setup
 - main.cpp, 75
- setup_inventory_and_scale
 - main.cpp, 75
- setup_rfid_and_lock
 - main.cpp, 75
- setup_RFID_reader
 - rfid_access.cpp, 81
 - rfid_access.h, 43
- setup_scale
 - weight_scale.cpp, 88
 - weight_scale.h, 52
- setup_web_routes
 - main.cpp, 75
- soda
 - inventory.h, 26
- src/buzzer.cpp, 54
- src/database_management.cpp, 57
- src/fridge_state.cpp, 61
- src/graph_data.cpp, 62
- src/init_users_and_sale.cpp, 64
- src/inventory.cpp, 65
- src/lock_ctrl.cpp, 69
- src/main.cpp, 73
- src/rfid_access.cpp, 77
- src/sale_html.cpp, 82
- src/weight_scale.cpp, 85
- SS_PIN
 - rfid_access.h, 36
- START_BEER_QTY
 - main.cpp, 76
- style_css.h
 - PROGMEM, 48
- tare_complete
 - weight_scale.cpp, 88
 - weight_scale.h, 53
- tare_scale
 - weight_scale.cpp, 88
 - weight_scale.h, 53
- TONE_LENGTH
 - buzzer.cpp, 56
 - lock_ctrl.cpp, 72
- uid
 - User, 12
- UID_LENGTH
 - rfid_access.h, 37
- unlock_door
 - lock_ctrl.cpp, 71
 - lock_ctrl.h, 32
- UNLOCK_POS
 - lock_ctrl.cpp, 72
- update_scale
 - weight_scale.cpp, 88
 - weight_scale.h, 53
- User, 11
 - balance, 12
 - roomNumber, 12
 - uid, 12
- user_management
 - database_management.cpp, 60
 - rfid_access.h, 44
- userCount
 - rfid_access.cpp, 82
 - rfid_access.h, 44
- users
 - rfid_access.cpp, 82
 - rfid_access.h, 44
- validate_rfid
 - rfid_access.cpp, 81
 - rfid_access.h, 44
- weight
 - product, 9
- weight_reference_is_set
 - weight_scale.cpp, 88
 - weight_scale.h, 53

- weight_scale.cpp
 - g_referenceWeight, [89](#)
 - get_beer_cans_taken, [86](#)
 - get_weight, [86](#)
 - get_weight_reference, [87](#)
 - reset_weight_reference, [87](#)
 - scale, [87](#)
 - set_weight_reference, [87](#)
 - setup_scale, [88](#)
 - tare_complete, [88](#)
 - tare_scale, [88](#)
 - update_scale, [88](#)
 - weight_reference_is_set, [88](#)
- weight_scale.h
 - BEER_WEIGHT, [50](#)
 - get_beer_cans_taken, [51](#)
 - get_weight, [51](#)
 - get_weight_reference, [52](#)
 - HX711_DOUT, [50](#)
 - HX711_SCK, [50](#)
 - reset_weight_reference, [52](#)
 - scale, [54](#)
 - SCALE_DEFAULT_SETTLE_TIME_MS, [51](#)
 - SCALE_TOL, [51](#)
 - set_weight_reference, [52](#)
 - setup_scale, [52](#)
 - tare_complete, [53](#)
 - tare_scale, [53](#)
 - update_scale, [53](#)
 - weight_reference_is_set, [53](#)
- WIFI_PASS
 - main.cpp, [77](#)
- WIFI_SSID
 - main.cpp, [77](#)