BFIT

Generated by Doxygen 1.9.1

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:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

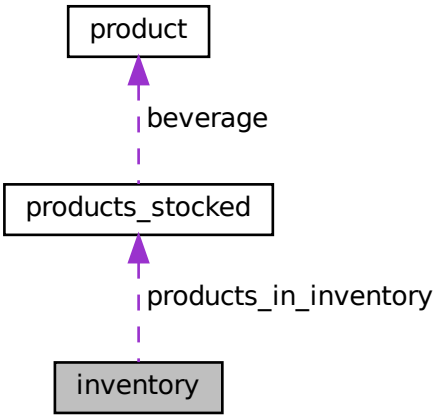
Class Documentation

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:

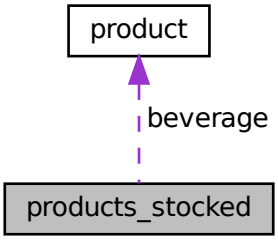
- include/inventory.h

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

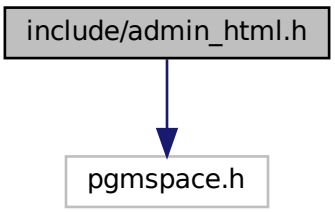
Chapter 5

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.

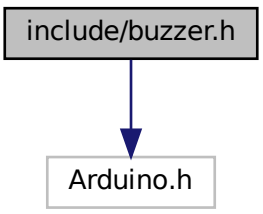
Opening container for the sales graph.

5.2 include/buzzer.h File Reference

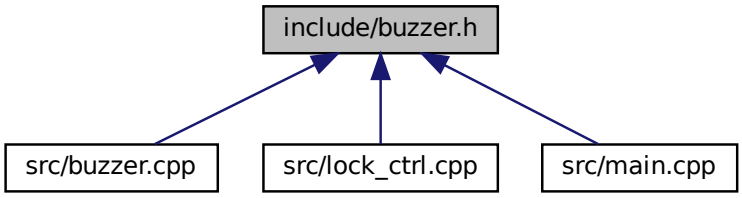
Implementation of buzzer sound effects.

```
#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

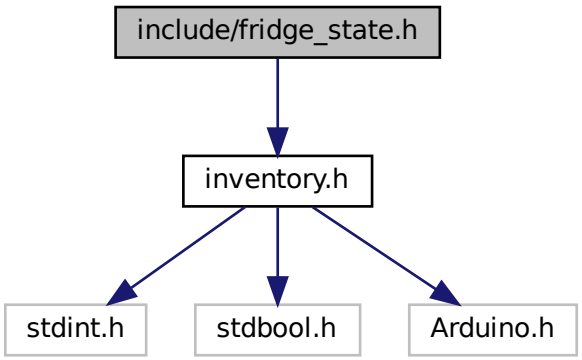
t	

5.3 include/fridge_state.h File Reference

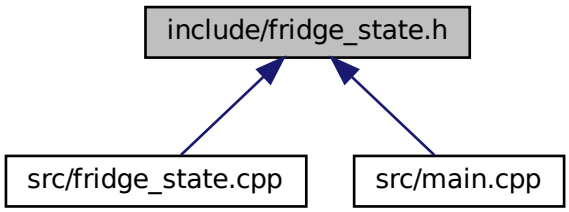
File for declearing the fridge inventory, user inventory shuld also be moved to this file, and the file renamed to reflekt the new content.

```
#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 declearing the fridge inventory, user inventory shuld also be moved to this file, and the file renamed to reflekt the new content.

Author

Baldur G. Toftegaard

5.3.2 Variable Documentation

5.3.2.1 fridge

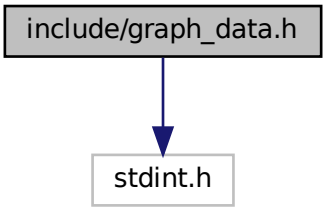
```
inventory fridge  [extern]
```

Used to set up the fridge inventory.

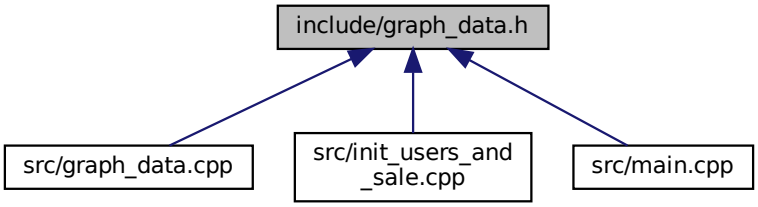
5.4 include/graph_data.h File Reference

```
#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

roomNumber	
delta	

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

roomNumber	
delta	

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.

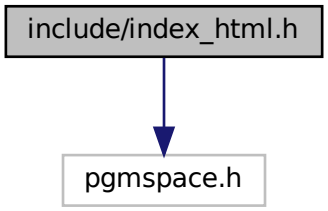
Indexed by room number. Used by sale_html and the /saleHeights endpoint.

5.5 include/index_html.h File Reference

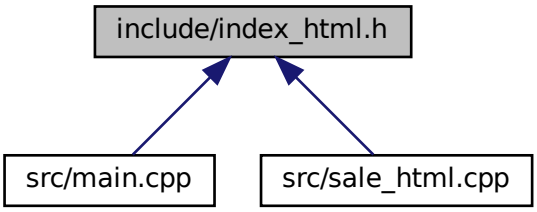
HTML for the start page.

```
#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

  Sting 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"
```

Sting used for the HTML header.

Opening container for the sales graph.

String used for the HTML footer.

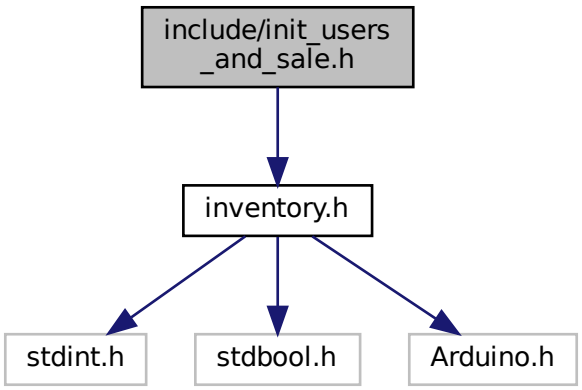
This is responsible for updating the graphs.

5.6 include/init_users_and_sale.h File Reference

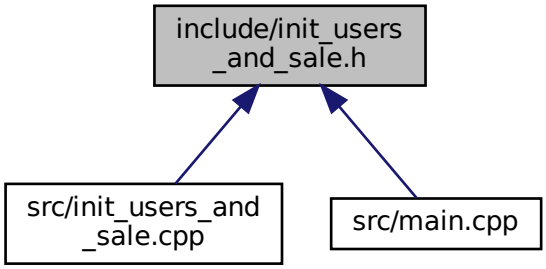
Function used for declearing the system users and products.

```
#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 declearing 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

weight	
user_id	
fridge_inventory	

Parameters

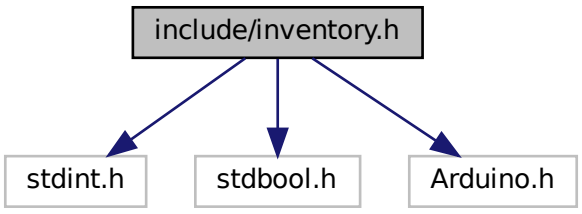
fridge_inventory	Inventory that the sale shuld remove item from

5.7 include/inventory.h File Reference

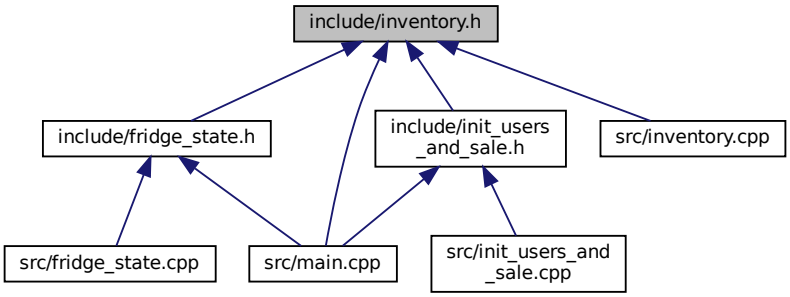
Inventory system for tracking the inventory of both the fridge and the induvidual users.

```
#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 induvidual 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

inventory	Inventory to modify.
beverage	Beverage to update.
amount	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

inventory	Inventory to add the product to.
product	Product to add.
quantity	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

inventory	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

name	Display name of the product.
type	Beverage type.
weight	Weight of the product.
price	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

inventory	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

inventory	Inventory to modify.
beverage	Beverage to update.
amount	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

inventory	Inventory to remove the product from.
beverage	Product to remove.

Returns

true Product removed successfully.

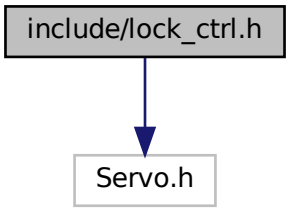
false Product not found.

5.8 include/lock_ctrl.h File Reference

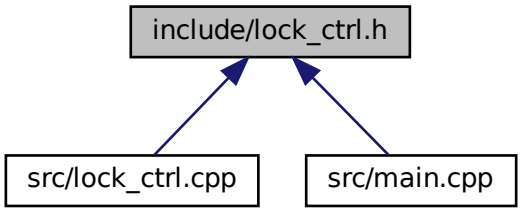
Door lock control interface using a servo and light sensor.

```
#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()

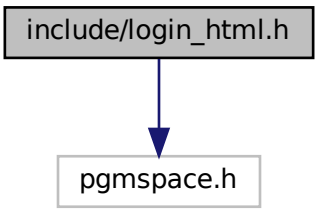
```
void unlock_door ( )
```

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.

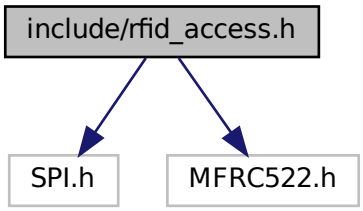
Opening container for the sales graph.

Interpreted as a string by the compiler.

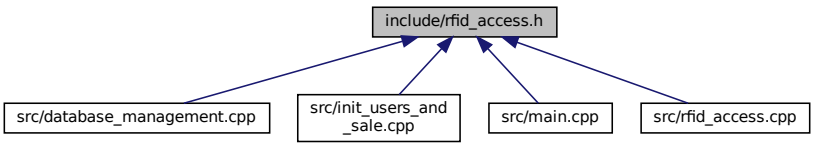
5.10 include/rfid_access.h File Reference

```
#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

rfid	

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 UIDs.

Parameters

uid1	
uid2	

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

ptr	

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

ptr	

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

ptr	

5.10.4.9 print_all_users()

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

Prints all users in the database to the serial interface.

Parameters

ptr	

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

ptr	
idx	

5.10.4.11 print_uid()

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

Prints a UID buffer to the serial interface.

Parameters

ptr	

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

rfid	
uidBuffer	

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

uidOut	

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

uidOut	

Returns

true

false

5.10.4.18 setup_RFID_reader()

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

Initializes SPI and the MFRC522 RFID reader.

Parameters

rfid	

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

cmd	
ptr	
rfid	

5.10.4.20 validate_rfid()

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

Validates an RFID tag against the registered user database.

Parameters

myRFID	

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]
```

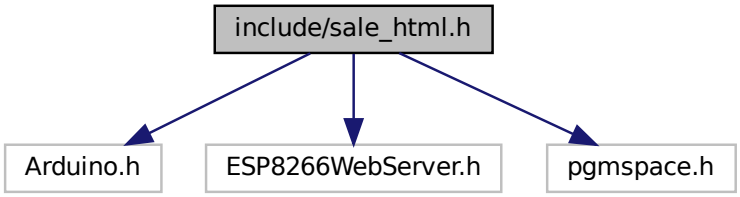
Global user database array.

5.11 include/sale_html.h File Reference

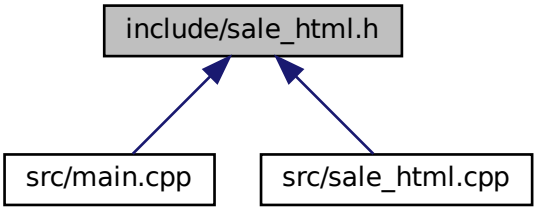
Headerfile for displaying the graph of sales on the main page.

```
#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

server	
room_number	
bar_type	
bar_height	

Parameters

server	The server
room_number	Number of the relevant room
bar_type	The type of the bar graph
bar_height	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

server	
room_count	
greenHeight	
classicHeights	

Parameters

server	The server
room_count	The number of rooms
greenHeight	The hight of the green bar
classicHeights	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.

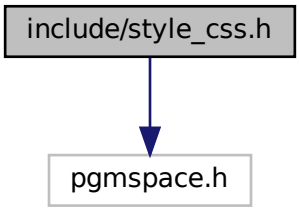
is handled like a string by the compiler

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.

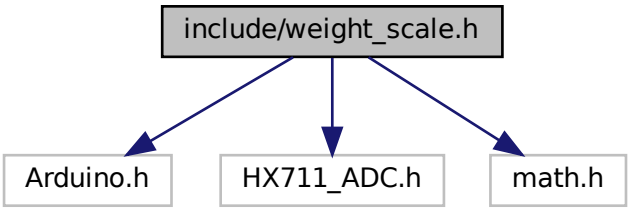
Opening container for the sales graph.

is handled like a string by the compiler

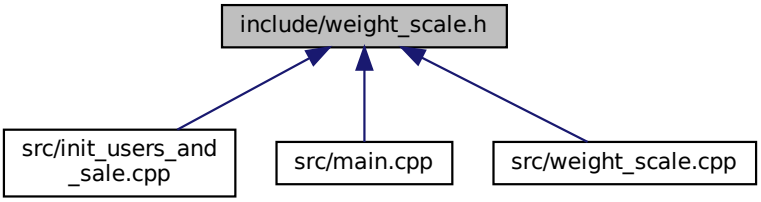
5.13 include/weight_scale.h File Reference

```
#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

referencWeight	
currentWeight	

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

value	

5.13.3.6 setup_scale()

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

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

Parameters

calFactor	

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

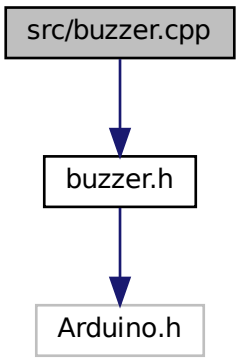
```
HX711_ADC scale  [extern]
```

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

t	

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
```

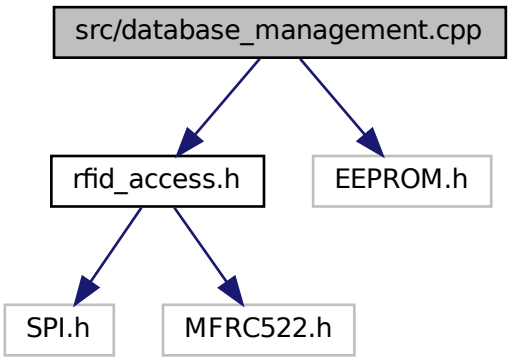
Duration of buzzer tone in milliseconds.

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

ptr	

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

ptr	

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

ptr	

5.16.2.4 print_all_users()

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

Prints all users in the database to the serial interface.

Parameters

ptr	

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

ptr	
idx	

5.16.2.6 print_uid()

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

Prints a UID buffer to the serial interface.

Parameters

ptr	

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.

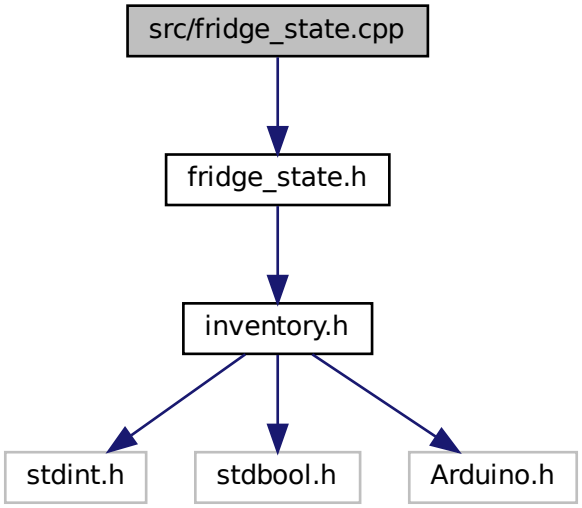
Parameters

cmd	
ptr	
rfid	

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

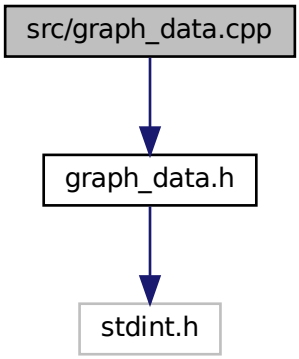
```
inventory fridge
```

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

roomNumber	
delta	

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

roomNumber	
delta	

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.

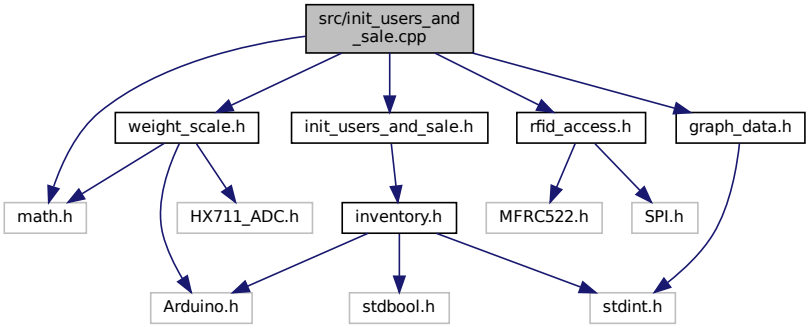
Height values for green product sales per room.

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, shuld 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, shuld 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

weight	
user_id	
fridge_inventory	

Parameters

fridge_inventory	Inventory that the sale shuld remove item from

5.19.1.3 read_current_weight_blocking()

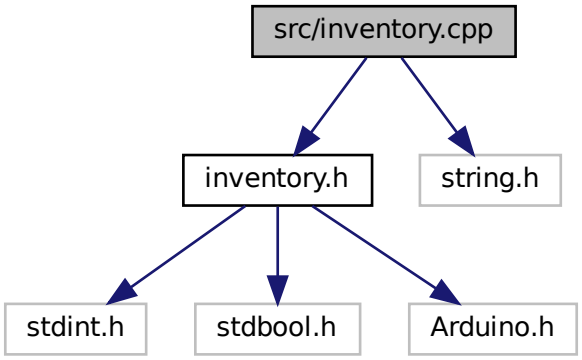
```
static float read_current_weight_blocking (
            uint32_t timeoutMs = 1200 )  [static]
```

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

inventory	Inventory to add the product to.
product	Product to add.
quantity	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

inventory	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

name	Display name of the product.
type	Beverage type.
weight	Weight of the product.
price	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

inventory	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

inventory	Inventory to modify.
beverage	Beverage to update.
amount	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

inventory	Inventory to remove the product from.
beverage	Product to remove.

Returns

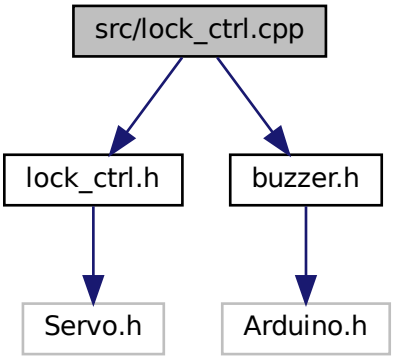
true Product removed successfully.

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]
```

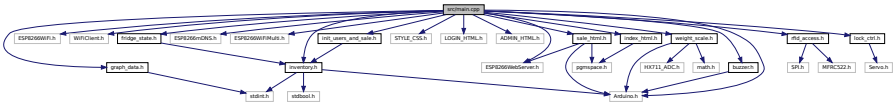
Servo position corresponding to the unlocked state (mechanically fixed).

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, 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, 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

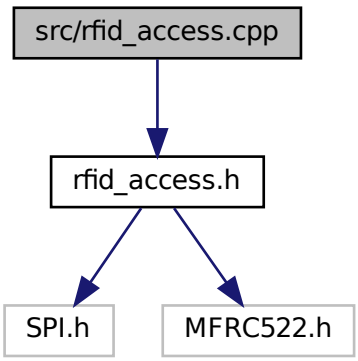
```
const char* WIFI_SSID = "Baldur's A56"
```

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 UIDs.
- bool read_RFID_tag (MFRC522 &rfid, byte ∗uidBuffer)

  Reads an RFID tag UID 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 [UID_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

rfid	

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 UIDs.

Parameters

uid1	
uid2	

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

rfid	
uidBuffer	

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

uidOut	

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

uidOut	

Returns

true

false

5.23.2.9 setup_RFID_reader()

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

Initializes SPI and the MFRC522 RFID reader.

Parameters

rfid	

5.23.2.10 validate_rfid()

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

Validates an RFID tag against the registered user database.

Parameters

myRFID	

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

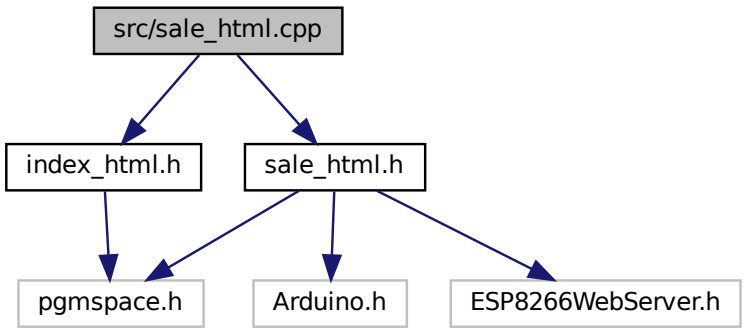
```
User users[MAX_ROOMS]
```

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( )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

server	
room_number	
bar_type	
bar_height	

Parameters

server	The server
room_number	Number of the relevant room
bar_type	The type of the bar graph
bar_height	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

server	
room_count	
greenHeight	
classicHeights	

Parameters

server	The server
room_count	The number of rooms
greenHeight	The hight of the green bar
classicHeights	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.

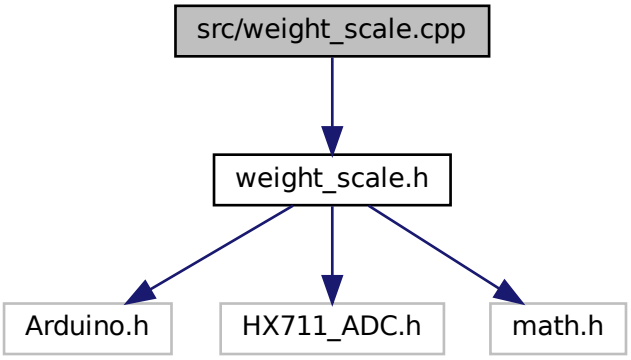
Interpreted as a string by the compiler.

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

referencWeight	
currentWeight	

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

value	

5.25.2.7 setup_scale()

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

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

Parameters

calFactor	

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