RTOS semestral project

Generated by Doxygen 1.8.17

Chapter 1

CTU_RTSP_term_project

assignment page link

1.0.1 Team

- Denys Datsko
- Morhunenko Mykola

1.1 Short project description

The main goal of the semester project was to create such a program, that reads the position from one motor (moved by hand or steer-by-wire), sends desired position via the Internet to another motor and sets it to another motor. The set-point is transferred between two motor controllers using UDP messages. The actual state of the controller and its history is published as live plots over http protocol.

We implemented:

- IRQ handler for reading and (maybe) updating some information
- motor driver for controlling step motors
- partially implemented PID controller for motors (P part, as we understand it, is implemented)
- UDP server for inter-controllers-communication using BSD sockets
- HTTP server for data visualisation in a browser
- SVG Plots library (we took `nanosvg` lib, modified it as we need, and wrote documentation + few more functions for it)

1.2 Instructions for compiling and running your application.

Project was written and tested only on Linux-based OS, for VxWorks RTOS in WindRiver IDE. Such files as `.project`, `.cpoject`, `.wrproject` and `.wrmakefile` files are given for building and executing.
So to compile the project, open it using the WindRiver IDE and build it **Project ->** **Build project**

1.3 Screenshot of your web-based (or text-based) user interface

TODO

1.4 Data-Flow Diagram.

1.5 Description of global functions and variables (created by you) including the description of function parameters and return values.

The documentation generation by Doxygen and `doxy.conf` is possible. You can generate `doc/` folder with `doxygen doxy.conf`. To receive `.pdf`, go to `/doc/latex` folder and run `make` there. To open the documentation as a web page, go to `doc/html/` folder and open the html file using any browser

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

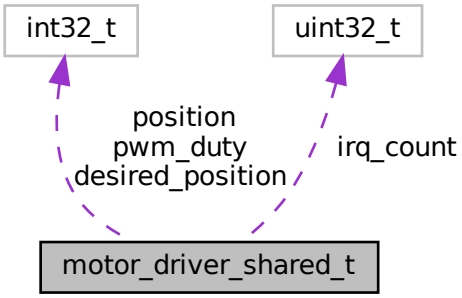
Data Structure Documentation

4.1 motor_driver_shared_t Struct Reference

Structure for storing shared data with motor driver.

`#include <shared.h>`

Collaboration diagram for motor_driver_shared_t:



Data Fields

- uint32_t irq_count
  Number of interrupts handled.
- int32_t position
  Absolute position of the motor.
- int32_t desired_position
  Desired position of the motor.
- int32_t pwm_duty
  Current PWM duty cycle.

4.1.1 Detailed Description

Structure for storing shared data with motor driver.

Definition at line 20 of file shared.h.

4.1.2 Field Documentation

4.1.2.1 desired_position

```
int32_t motor_driver_shared_t::desired_position
```

Desired position of the motor.

Definition at line 23 of file shared.h.

Referenced by motor(), motor_isr(), run_udp_srv(), and update_statistics().

4.1.2.2 irq_count

```
uint32_t motor_driver_shared_t::irq_count
```

Number of interrupts handled.

Definition at line 21 of file shared.h.

Referenced by motor_init(), and motor_isr().

4.1.2.3 position

```
int32_t motor_driver_shared_t::position
```

Absolute position of the motor.

Definition at line 22 of file shared.h.

Referenced by motor(), motor_init(), motor_isr(), run_udp_srv(), and update_statistics().

4.1.2.4 pwm_duty

```
int32_t motor_driver_shared_t::pwm_duty
```

Current PWM duty cycle.

Definition at line 24 of file shared.h.

Referenced by motor(), motor_isr(), and update_statistics().

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

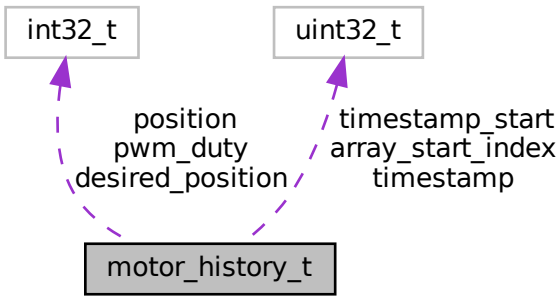
- include/shared.h

4.2 motor_history_t Struct Reference

Structure for storing the hostiry of motor_driver_shared_t.

```
#include <shared.h>
```

Collaboration diagram for motor_history_t:



Data Fields

- int32_t position [1200]
  cyclic array storing absolute position of the motor
- int32_t desired_position [1200]
  cyclic array storing desired position of the motor
- int32_t pwm_duty [1200]
  cyclic array storing PWM duty cycle hitory
- uint32_t timestamp [1200]
  array storing corresponding timestamps
- uint32_t timestamp_start
  Application start timestamp.
- uint32_t array_start_index
  Start index in cyclic arrays of this structure.

4.2.1 Detailed Description

Structure for storing the hostiry of motor_driver_shared_t.

Definition at line 29 of file shared.h.

4.2.2 Field Documentation

4.2.2.1 array_start_index

```
uint32_t motor_history_t::array_start_index
```

Start index in cyclic arrays of this structure.

Definition at line 35 of file shared.h.

Referenced by generate_html_file(), and update_statistics().

4.2.2.2 desired_position

```
int32_t motor_history_t::desired_position[1200]
```

cyclic array storing desired position of the motor

Definition at line 31 of file shared.h.

Referenced by create_tasks(), generate_html_file(), and update_statistics().

4.2.2.3 position

```
int32_t motor_history_t::position[1200]
```

cyclic array storing absolute position of the motor

Definition at line 30 of file shared.h.

Referenced by create_tasks(), generate_html_file(), and update_statistics().

4.2.2.4 pwm_duty

`int32_t motor_history_t::pwm_duty[1200]`

cyclic array storing PWM duty cycle hitory

Definition at line 32 of file shared.h.

Referenced by create_tasks(), generate_html_file(), and update_statistics().

4.2.2.5 timestamp

`uint32_t motor_history_t::timestamp[1200]`

array storing corresponding timestamps

Definition at line 33 of file shared.h.

Referenced by generate_html_file(), and update_statistics().

4.2.2.6 timestamp_start

`uint32_t motor_history_t::timestamp_start`

Application start timestamp.

Definition at line 34 of file shared.h.

Referenced by create_tasks().

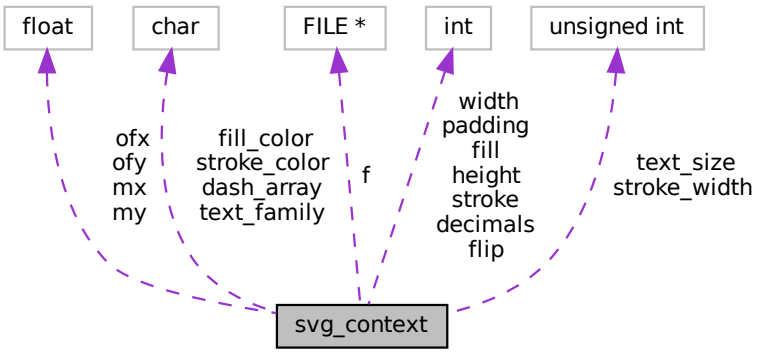
The documentation for this struct was generated from the following file:

- include/shared.h

4.3 svg_context Struct Reference

`#include <svg_generator.h>`

Collaboration diagram for svg_context:



Data Fields

- char stroke_color [20]
- char fill_color [20]
- char text_family [30]
- char dash_array [30]
- unsigned int text_size
- unsigned int stroke_width
- int stroke
- int fill
- int padding
- int width
- int height
- int flip
- int decimals
- float mx
- float my
- float ofx
- float ofy
- FILE * f

4.3.1 Detailed Description

Definition at line 17 of file svg_generator.h.

4.3.2 Field Documentation

4.3.2.1 dash_array

```
char svg_context::dash_array[30]
```

Definition at line 21 of file svg_generator.h.

Referenced by svg_draw_to_file_xy(), svg_opt_stroke(), and svg_set_dash().

4.3.2.2 decimals

```
int svg_context::decimals
```

Definition at line 30 of file svg_generator.h.

4.3.2.3 f

```
FILE* svg_context::f
```

Definition at line 32 of file svg_generator.h.

Referenced by svg_draw_to_file_xy(), svg_finish(), svg_opt_end(), svg_opt_fill(), svg_opt_fill_none(), svg_opt_long_beg(), svg_opt_long_end(), svg_opt_stroke(), svg_opta(), svg_optf(), svg_opti(), svg_path_data(), svg_rotate(), svg_scale(), svg_tag_end(), svg_tag_start(), svg_text(), and svg_translate().

4.3.2.4 fill

```
int svg_context::fill
```

Definition at line 25 of file svg_generator.h.

4.3.2.5 fill_color

```
char svg_context::fill_color[20]
```

Definition at line 19 of file svg_generator.h.

Referenced by svg_draw_to_file_xy(), svg_opt_fill(), and svg_set_fill().

4.3.2.6 flip

```
int svg_context::flip
```

Definition at line 29 of file svg_generator.h.

4.3.2.7 height

```
int svg_context::height
```

Definition at line 28 of file svg_generator.h.

Referenced by svg_draw_to_file_xy().

4.3.2.8 mx

```
float svg_context::mx
```

Definition at line 31 of file svg_generator.h.

Referenced by svg_draw_to_file_xy().

4.3.2.9 my

```
float svg_context::my
```

Definition at line 31 of file svg_generator.h.

Referenced by svg_draw_to_file_xy().

4.3.2.10 ofx

```
float svg_context::ofx
```

Definition at line 31 of file svg_generator.h.

Referenced by svg_draw_to_file_xy().

4.3.2.11 ofy

```
float svg_context::ofy
```

Definition at line 31 of file svg_generator.h.

Referenced by svg_draw_to_file_xy().

4.3.2.12 padding

```
int svg_context::padding
```

Definition at line 27 of file svg_generator.h.

Referenced by svg_draw_to_file_xy().

4.3.2.13 stroke

```
int svg_context::stroke
```

Definition at line 25 of file svg_generator.h.

4.3.2.14 stroke_color

```
char svg_context::stroke_color[20]
```

Definition at line 18 of file svg_generator.h.

Referenced by svg_draw_to_file_xy(), svg_opt_stroke(), and svg_set_stroke().

4.3.2.15 stroke_width

```
unsigned int svg_context::stroke_width
```

Definition at line 23 of file svg_generator.h.

Referenced by svg_draw_to_file_xy(), and svg_opt_stroke().

4.3.2.16 text_family

```
char svg_context::text_family[30]
```

Definition at line 20 of file svg_generator.h.

Referenced by svg_draw_to_file_xy(), and svg_text().

4.3.2.17 text_size

```
unsigned int svg_context::text_size
```

Definition at line 22 of file svg_generator.h.

Referenced by svg_draw_to_file_xy(), and svg_text().

4.3.2.18 width

```
int svg_context::width
```

Definition at line 28 of file svg_generator.h.

Referenced by svg_draw_to_file_xy().

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

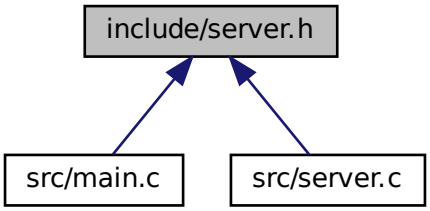
- include/svg_generator.h

Chapter 5

File Documentation

5.1 include/server.h File Reference

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



Macros

- #define DEFAULT_PORT 8000

Functions

- void run_udp_srv (char *ip_address, int port)

  Run udp server for sharing data We use this function for communication between two microzed boards It sends desire position from one board to another using BSD sockets.

5.1.1 Macro Definition Documentation

5.1.1.1 DEFAULT_PORT

```
#define DEFAULT_PORT 8000
```

Definition at line 4 of file server.h.

5.1.2 Function Documentation

5.1.2.1 run_udp_srv()

```
void run_udp_srv (
            char * ip_address,
            int port )
```

Run udp server for sharing data We use this function for communication between two microzed boards It sends desire position from one board to another using BSD sockets.

Parameters

ip_address	: char* - server's ip address
port	: int - udp port for sockets

Definition at line 25 of file server.c.

```
25                                          {
26     // init semaphore
27     SEM_ID sem_motor_pwm_change = semOpen("/sem_motor_pwm_change", SEM_TYPE_COUNTING, SEM_EMPTY,
       SEM_Q_FIFO, OM_CREATE, NULL);
28
29     // information_preprocessing
30     if (sysTimestampEnable() == ERROR) {
31         perror("Time stamps could not be enabled");
32         return;
33     }
34
35     int sockd;
36     struct sockaddr_in my_name, cli_name, srv_addr;
37     socklen_t addrlen;
38     const int buf_len = sizeof(int32_t);
39     int i, status;
40
41     // Common part for both receiver and the transmitter
42     sockd = socket(AF_INET, SOCK_DGRAM, 0);
43     if (sockd == -1) {
44         perror("Could not open the socket...");
45         exit(1);
46     }
47
48     my_name.sin_family = AF_INET;
49     my_name.sin_addr.s_addr = INADDR_ANY;
50     my_name.sin_port = htons(port);
51
52     if (bind(sockd, (struct sockaddr *) &my_name, sizeof(my_name)) == -1) {
53         perror("Could not bind the socket...");
54         close(sockd);
55         exit(2);
56     }
57
58     addrlen = sizeof(cli_name);
59
60
61     if (ip_address == NULL) {
62         // Running as a receiver
63         //      set int32_t motor_driver_shared.desired_position
64 #ifdef DEBUG_OUTPUT
```

```
65          printf("Runnign as a receiver...\n");
66  #endif
67
68          while (1) {
69              status = recvfrom(sockd, &(motor_driver_shared.desired_position), buf_len, 0, (struct
        sockaddr *) &cli_name, & addrlen);
70              semGive(sem_motor_pwm_change);
71  #ifdef DEBUG_OUTPUT
72              printf("received %i\n", motor_driver_shared.desired_position);
73  #endif
74          }
75          close(sockd);
76      } else {
77  #ifdef DEBUG_OUTPUT
78          printf("Runnign as a transmitter...\n");
79  #endif
80          // Running as a transmitter
81          //      transfer int32_t motor_driver_shared.desired_position
82          srv_addr.sin_family = AF_INET;
83          inet_aton(ip_address, &srv_addr.sin_addr);
84          srv_addr.sin_port = htons(port);
85
86          i = 0;
87          while (1) {
88              sendto(sockd, &(motor_driver_shared.position), buf_len, 0, (struct sockaddr *) &srv_addr,
        sizeof(srv_addr));
89  #ifdef DEBUG_OUTPUT
90              printf("sent %i\n", motor_driver_shared.position);
91  #endif
92              taskDelay(SERVER_SEND_DELAY);
93          }
94          close(sockd);
95      }
96      printf("Measurement finished\n");
97  }
```

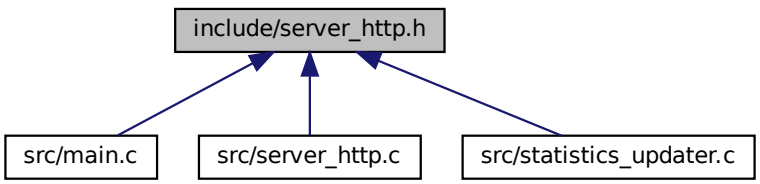
References motor_driver_shared_t::desired_position, motor_driver_shared, motor_driver_shared_t::position, sem_motor_pwm_change, and SERVER_SEND_DELAY.

Referenced by create_tasks().

Here is the caller graph for this function:

5.2 include/server_http.h File Reference

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



Macros

- #define HTTP_SERVER_PORT 80 /* Port 80 is reserved for HTTP protocol */
- #define HTTP_SERVER_MAX_CONNECTIONS 20
- #define HTTP_RECCEIVER_BUF_SIZE 5000

Functions

- void run_http_srv ()
- void update_statistics ()

5.2.1 Macro Definition Documentation

5.2.1.1 HTTP_RECCEIVER_BUF_SIZE

```
#define HTTP_RECCEIVER_BUF_SIZE 5000
```

Definition at line 6 of file server_http.h.

5.2.1.2 HTTP_SERVER_MAX_CONNECTIONS

```
#define HTTP_SERVER_MAX_CONNECTIONS 20
```

Definition at line 5 of file server_http.h.

5.2.1.3 HTTP_SERVER_PORT

```
#define HTTP_SERVER_PORT 80 /* Port 80 is reserved for HTTP protocol */
```

Definition at line 4 of file server_http.h.

5.2.2 Function Documentation

5.2.2.1 run_http_srv()

```
void run_http_srv ( )
```

Entry point of the task for running the HTTP server The task creates a new task for serving each request from any client

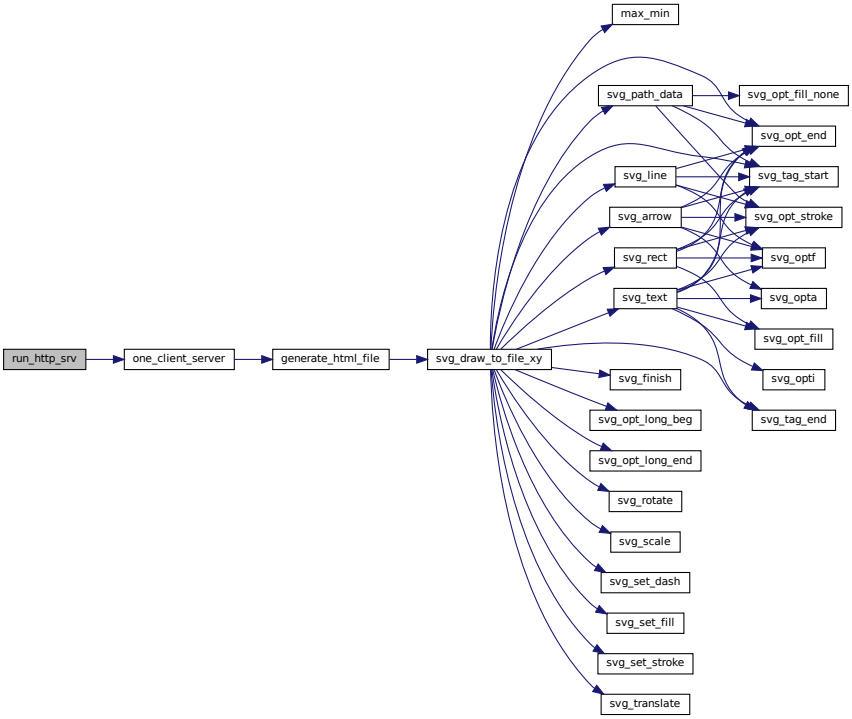
Definition at line 70 of file server_http.c.

```
70                      {
71     int s;
72     int new_fd;
73     struct sockaddr_in serverAddr;
74     struct sockaddr_in clientAddr;
75     socklen_t sockAddrSize;
76 
77     sockAddrSize = sizeof(struct sockaddr_in);
78     bzero((char *) &serverAddr, sizeof(struct sockaddr_in));
79     serverAddr.sin_family = AF_INET;
80     serverAddr.sin_port = htons(HTTP_SERVER_PORT);
81     serverAddr.sin_addr.s_addr = htonl(INADDR_ANY);
82 
83     s = socket(AF_INET, SOCK_STREAM, 0);
84     if (s < 0) {
85         printf("Error: www: socket(%d)\n", s);
86         return;
87     }
88 
89 
90     if (bind(s, (struct sockaddr *) &serverAddr, sockAddrSize) == ERROR) {
91         printf("Error: www: bind\n");
92         return;
93     }
94 
95     if (listen(s, HTTP_SERVER_MAX_CONNECTIONS) == ERROR) {
96         perror("www listen");
97         close(s);
98         return;
99     }
100 
101 #ifdef DEBUG_OUTPUT
102     printf("www server running\n");
103 #endif
104 
105     char task_name_buf[100];
106 
107     int client_num = 0;
108     while (1) {
109         /* accept waits for somebody to connect and the returns a new file descriptor */
110         if ((new_fd = accept(s, (struct sockaddr *) &clientAddr, &sockAddrSize)) == ERROR) {
111             perror("www accept");
112             close(s);
113             return;
114         }
115         sprintf(task_name_buf, "tOneClientServer%d", client_num++);
116         if (taskSpawn(task_name_buf, HTTP_SERVER_PRIORITY, VX_FP_TASK, 65536, (FUNCPTR)
       one_client_server, new_fd, 0, 0, 0, 0, 0, 0, 0, 0) == ERROR) {
117             perror("Creating task for communication with an client");
118             close(s);
119             close(new_fd);
120             return;
121         }
122     }
123 }
```

References HTTP_SERVER_MAX_CONNECTIONS, HTTP_SERVER_PORT, HTTP_SERVER_PRIORITY, and one_client_server().

Referenced by create_tasks().

Here is the call graph for this function:



Here is the caller graph for this function:

5.2.2.2 update_statistics()

```
void update_statistics ( )
```

Entry point of task that contonuously updates motor history statistics

Written using Ostrich algorithm - we ignore data race in this function Obviously it can happen (somewhere on the bounds) and we realise it But to use mutex/semaphore here will be a huge overhead for the whole system.

Definition at line 15 of file statistics_updater.c.

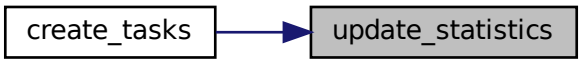
```
15                          {
16     uint32_t prev_timestamp = sysTimestamp(), cur_timestamp;
```

```
17      uint64_t timestamp64 = 0;
18
19      while (1) {
20          cur_timestamp = sysTimestamp();
21          timestamp64 += cur_timestamp - prev_timestamp;
22          prev_timestamp = cur_timestamp;
23
24          uint32_t i = (motor_history.array_start_index + HISTORY_SIZE) % HISTORY_CYCLIC_ARRAY_SIZE;
25          motor_history.desired_position[i] = motor_driver_shared.desired_position;
26          motor_history.position[i] = motor_driver_shared.position;
27          motor_history.pwm_duty[i] = motor_driver_shared.pwm_duty;
28          motor_history.timestamp[i] = (timestamp64 * 1000) / sysTimestampFreq();
29
30          motor_history.array_start_index = (motor_history.array_start_index + 1) %
        HISTORY_CYCLIC_ARRAY_SIZE;
31          taskDelay(3);
32      }
33  }
```

References motor_history_t::array_start_index, motor_driver_shared_t::desired_position, motor_history_t::desired_position, HISTORY_CYCLIC_ARRAY_SIZE, HISTORY_SIZE, motor_driver_shared, motor_history, motor_driver_shared_t::position, motor_history_t::position, motor_driver_shared_t::pwm_duty, motor_history_t::pwm_duty, and motor_history_t::timestamp.

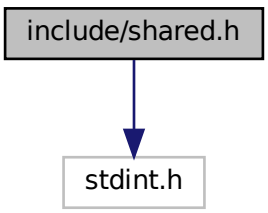
Referenced by create_tasks().

Here is the caller graph for this function:

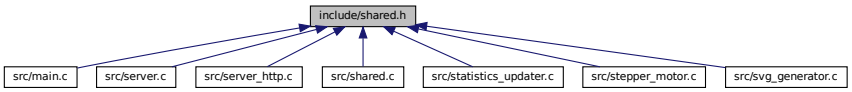


5.3 include/shared.h File Reference

`#include <stdint.h>`
Include dependency graph for shared.h:



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



Data Structures

- struct motor_driver_shared_t

  Structure for storing shared data with motor driver.
- struct motor_history_t

  Structure for storing the hostiry of motor_driver_shared_t.

Macros

- #define SERVER_SEND_DELAY 50
- #define CLOCK_RATE 1000
- #define MOTOR_DRIVER_PRIOROTY 150
- #define UDP_COMMUNICATION_PRIORITY 151
- #define HTTP_SERVER_PRIORITY 153
- #define STATISTICS_MONITOR_PRIORITY 152
- #define HISTORY_SIZE 800

  Number of values in cyclic arrays (without gap)
- #define HISTORY_CYCLIC_ARRAY_SIZE 1200

  Actual size of cyclic arrays. Should be greater than HISTORY_SIZE.
- #define DEBUG_OUTPUT

Variables

- motor_driver_shared_t motor_driver_shared
- motor_history_t motor_history

5.3.1 Macro Definition Documentation

5.3.1.1 CLOCK_RATE

```
#define CLOCK_RATE 1000
```

Definition at line 7 of file shared.h.

5.3.1.2 DEBUG_OUTPUT

```
#define DEBUG_OUTPUT
```

Definition at line 41 of file shared.h.

5.3.1.3 HISTORY_CYCLIC_ARRAY_SIZE

```
#define HISTORY_CYCLIC_ARRAY_SIZE 1200
```

Actual size of cyclic arrays. Should be greater than HISTORY_SIZE.

Definition at line 16 of file shared.h.

5.3.1.4 HISTORY_SIZE

```
#define HISTORY_SIZE 800
```

Number of values in cyclic arrays (without gap)

Definition at line 14 of file shared.h.

5.3.1.5 HTTP_SERVER_PRIORITY

```
#define HTTP_SERVER_PRIORITY 153
```

Definition at line 10 of file shared.h.

5.3.1.6 MOTOR_DRIVER_PRIOROTY

```
#define MOTOR_DRIVER_PRIOROTY 150
```

Definition at line 8 of file shared.h.

5.3.1.7 SERVER_SEND_DELAY

```
#define SERVER_SEND_DELAY 50
```

Definition at line 6 of file shared.h.

5.3.1.8 STATISTICS_MONITOR_PRIORITY

```
#define STATISTICS_MONITOR_PRIORITY 152
```

Definition at line 11 of file shared.h.

5.3.1.9 UDP_COMMUNICATION_PRIORITY

```
#define UDP_COMMUNICATION_PRIORITY 151
```

Definition at line 9 of file shared.h.

5.3.2 Variable Documentation

5.3.2.1 motor_driver_shared

```
motor_driver_shared_t motor_driver_shared
```

Definition at line 3 of file shared.c.

Referenced by motor(), motor_init(), motor_isr(), run_udp_srv(), and update_statistics().

5.3.2.2 motor_history

```
motor_history_t motor_history
```

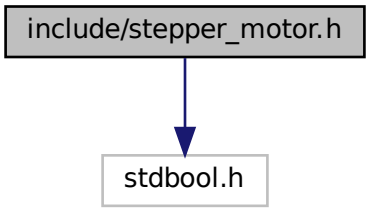
Definition at line 10 of file shared.c.

Referenced by create_tasks(), generate_html_file(), and update_statistics().

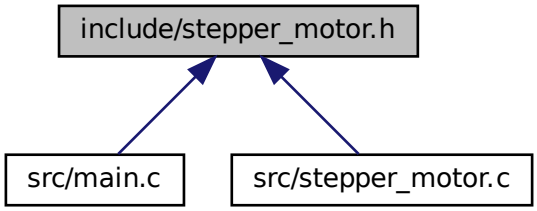
5.4 include/stepper_motor.h File Reference

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

Include dependency graph for stepper_motor.h:



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



Macros

- #define REGISTER(base, offs) (∗((volatile UINT32 ∗)((base) + (offs))))
- #define BIT(i) ((1) << (i))
- #define PWM_PERIOD 5000
- #define MIN_PWM_DUTY 3
- #define PMOD1_BASE 0x43c20000
- #define PMOD2_BASE 0x43c30000
- #define MOTOR_BASE PMOD1_BASE
- #define MOTOR_IRQ_PIN BIT(2)
- #define GPIO_DIRM REGISTER(ZYNQ7K_GPIO_BASE, 0x00000284)
- #define GPIO_INT_ENABLE REGISTER(ZYNQ7K_GPIO_BASE, 0x00000290)
- #define GPIO_INT_DISABLE REGISTER(ZYNQ7K_GPIO_BASE, 0x00000294)
- #define GPIO_INT_STATUS REGISTER(ZYNQ7K_GPIO_BASE, 0x00000298)
- #define GPIO_INT_TYPE REGISTER(ZYNQ7K_GPIO_BASE, 0x0000029c)
- #define GPIO_INT_POLARITY REGISTER(ZYNQ7K_GPIO_BASE, 0x000002a0)
- #define GPIO_INT_ANY REGISTER(ZYNQ7K_GPIO_BASE, 0x000002a4)

- #define MOTOR_SR REGISTER(MOTOR_BASE, 0x4)
- #define MOTOR_SR_IRC_A_MON 8
- #define MOTOR_SR_IRC_B_MON 9
- #define MOTOR_CR REGISTER(MOTOR_BASE, 0x0)
- #define MOTOR_CR_PWM_ENABLE 6
- #define MOTOR_CR_PWM_R_DIRECT 5
- #define MOTOR_CR_PWM_F_DIRECT 4
- #define PWM_PERIOD_REGISTER REGISTER(MOTOR_BASE, 0x8)
- #define PWM_DUTY_CR REGISTER(MOTOR_BASE, 0xC)
- #define DUTY_DIR_R 31
- #define DUTY_DIR_F 30
- #define DUTY_CYCLE 0

Functions

- void motor (bool set_desired)

  Entry point to the DKM.

5.4.1 Macro Definition Documentation

5.4.1.1 BIT

```
#define BIT(
              i ) ((1) << (i))
```

Definition at line 7 of file stepper_motor.h.

5.4.1.2 DUTY_CYCLE

```
#define DUTY_CYCLE 0
```

Definition at line 71 of file stepper_motor.h.

5.4.1.3 DUTY_DIR_F

```
#define DUTY_DIR_F 30
```

Definition at line 69 of file stepper_motor.h.

5.4.1.4 DUTY_DIR_R

```
#define DUTY_DIR_R 31
```

Definition at line 68 of file stepper_motor.h.

5.4.1.5 GPIO_DIRM

```
#define GPIO_DIRM REGISTER(ZYNQ7K_GPIO_BASE, 0x00000284)
```

Definition at line 33 of file stepper_motor.h.

5.4.1.6 GPIO_INT_ANY

```
#define GPIO_INT_ANY REGISTER(ZYNQ7K_GPIO_BASE, 0x000002a4)
```

Definition at line 53 of file stepper_motor.h.

5.4.1.7 GPIO_INT_DISABLE

```
#define GPIO_INT_DISABLE REGISTER(ZYNQ7K_GPIO_BASE, 0x00000294)
```

Definition at line 39 of file stepper_motor.h.

5.4.1.8 GPIO_INT_ENABLE

```
#define GPIO_INT_ENABLE REGISTER(ZYNQ7K_GPIO_BASE, 0x00000290)
```

Definition at line 36 of file stepper_motor.h.

5.4.1.9 GPIO_INT_POLARITY

```
#define GPIO_INT_POLARITY REGISTER(ZYNQ7K_GPIO_BASE, 0x000002a0)
```

Definition at line 50 of file stepper_motor.h.

5.4.1.10 GPIO_INT_STATUS

```
#define GPIO_INT_STATUS REGISTER(ZYNQ7K_GPIO_BASE, 0x00000298)
```

Definition at line 43 of file stepper_motor.h.

5.4.1.11 GPIO_INT_TYPE

```
#define GPIO_INT_TYPE REGISTER(ZYNQ7K_GPIO_BASE, 0x0000029c)
```

Definition at line 46 of file stepper_motor.h.

5.4.1.12 MIN_PWM_DUTY

```
#define MIN_PWM_DUTY 3
```

Definition at line 9 of file stepper_motor.h.

5.4.1.13 MOTOR_BASE

```
#define MOTOR_BASE PMOD1_BASE
```

Definition at line 17 of file stepper_motor.h.

5.4.1.14 MOTOR_CR

```
#define MOTOR_CR REGISTER(MOTOR_BASE, 0x0)
```

Definition at line 60 of file stepper_motor.h.

5.4.1.15 MOTOR_CR_PWM_ENABLE

```
#define MOTOR_CR_PWM_ENABLE 6
```

Definition at line 61 of file stepper_motor.h.

5.4.1.16 MOTOR_CR_PWM_F_DIRECT

```
#define MOTOR_CR_PWM_F_DIRECT 4
```

Definition at line 63 of file stepper_motor.h.

5.4.1.17 MOTOR_CR_PWM_R_DIRECT

```
#define MOTOR_CR_PWM_R_DIRECT 5
```

Definition at line 62 of file stepper_motor.h.

5.4.1.18 MOTOR_IRQ_PIN

```
#define MOTOR_IRQ_PIN BIT(2)
```

Definition at line 26 of file stepper_motor.h.

5.4.1.19 MOTOR_SR

```
#define MOTOR_SR REGISTER(MOTOR_BASE, 0x4)
```

Definition at line 56 of file stepper_motor.h.

5.4.1.20 MOTOR_SR_IRC_A_MON

```
#define MOTOR_SR_IRC_A_MON 8
```

Definition at line 57 of file stepper_motor.h.

5.4.1.21 MOTOR_SR_IRC_B_MON

```
#define MOTOR_SR_IRC_B_MON 9
```

Definition at line 58 of file stepper_motor.h.

5.4.1.22 PMOD1_BASE

```
#define PMOD1_BASE 0x43c20000
```

Definition at line 14 of file stepper_motor.h.

5.4.1.23 PMOD2_BASE

```
#define PMOD2_BASE 0x43c30000
```

Definition at line 15 of file stepper_motor.h.

5.4.1.24 PWM_DUTY_CR

```
#define PWM_DUTY_CR REGISTER(MOTOR_BASE, 0xC)
```

Definition at line 67 of file stepper_motor.h.

5.4.1.25 PWM_PERIOD

```
#define PWM_PERIOD 5000
```

Definition at line 8 of file stepper_motor.h.

5.4.1.26 PWM_PERIOD_REGISTER

```
#define PWM_PERIOD_REGISTER REGISTER(MOTOR_BASE, 0x8)
```

Definition at line 65 of file stepper_motor.h.

5.4.1.27 REGISTER

```
#define REGISTER(
              base,
              offs ) (*((volatile UINT32 *)((base) + (offs))))
```

Definition at line 6 of file stepper_motor.h.

5.4.2 Function Documentation

5.4.2.1 motor()

```
void motor (
             bool set_desired )
```

Entry point to the DKM.

Parameters

set_desired	: true if the driver should set position of the motor and not just monitor values

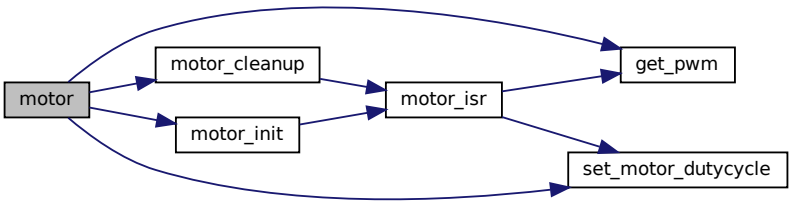
Definition at line 134 of file stepper_motor.c.

```
134                             {
135     set_position_to_desired = set_desired;
136     // Init semaphore for changing PWM duty requests
137     sem_motor_pwm_change = semOpen("/sem_motor_pwm_change", SEM_TYPE_COUNTING, SEM_EMPTY, SEM_Q_FIFO,
        OM_CREATE, NULL);
138 
139     motor_init();
140 
141     while (1) {
142         // Wait for the request to recalculate the duty cycle
143         semTake(sem_motor_pwm_change, WAIT_FOREVER);
144         if (set_desired) {
145             // Disable interrupts to prevent data race (when interrupt handler sets duty cycle too)
146             int8_t duty = get_pwm(motor_driver_shared.desired_position - motor_driver_shared.position);
147             intDisable(INT_LVL_GPIO);
148             set_motor_dutycycle(duty);
149             motor_driver_shared.pwm_duty = duty;
150             intEnable(INT_LVL_GPIO);
151         }
152     }
153 
154     semClose(sem_motor_pwm_change);
155     motor_cleanup();
156 }
```

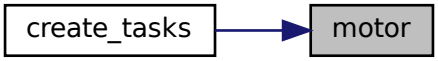
References motor_driver_shared_t::desired_position, get_pwm(), motor_cleanup(), motor_driver_shared, motor_↩ init(), motor_driver_shared_t::position, motor_driver_shared_t::pwm_duty, sem_motor_pwm_change, set_motor↩ _dutycycle(), and set_position_to_desired.

Referenced by create_tasks().

Here is the call graph for this function:



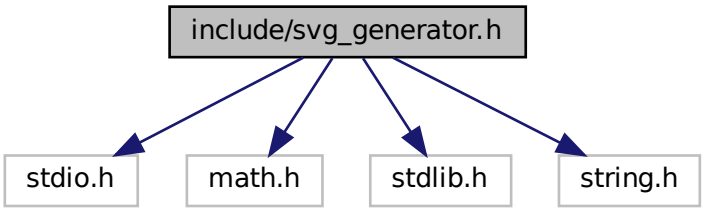
Here is the caller graph for this function:



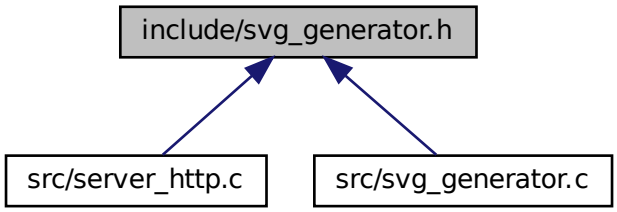
5.5 include/svg_generator.h File Reference

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

Include dependency graph for svg_generator.h:



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



Data Structures

- struct svg_context

Macros

- #define PLOT_WIDTH 1000
- #define PLOT_HEIGHT 300
- #define MARKER_ARROW fprintf(f,"<marker id=\"markerArrow\" markerWidth=\"13\" markerHeight=\"13\" refX=\"7\" refY=\"6\" orient=\"auto\">\n<path d=\"M2,8 L11,6 L2,4 Z\" style=\"fill: #000000;stroke: black;\" /> </marker>")
- #define cordx(var) (((var)-ctx->ofx)*(ctx->mx))
- #define cordy(var) (((var)-ctx->ofy)*(ctx->my))

Typedefs

- typedef struct svg_context svg_context

Functions

- void svg_draw_to_file_xy (FILE *f, const char *color, float *x, float *y, size_t count, const char *x_label, const char *y_label)
- void generate_html_file (FILE *f)

  Write html + svg plots to the file.

5.5.1 Macro Definition Documentation

5.5.1.1 cordx

```
#define cordx(
              var ) (((var)-ctx->ofx)*(ctx->mx))
```

Definition at line 14 of file svg_generator.h.

5.5.1.2 cordy

```
#define cordy(
              var ) (((var)-ctx->ofy)*(ctx->my))
```

Definition at line 15 of file svg_generator.h.

5.5.1.3 MARKER_ARROW

```
#define MARKER_ARROW fprintf(f,"<marker id=\"markerArrow\" markerWidth=\"13\" markerHeight=\"13\"
refX=\"7\" refY=\"6\" orient=\"auto\">\n<path d=\"M2,8 L11,6 L2,4 Z\" style=\"fill:  #000000;stroke↩
:  black;\" /> </marker>")
```

Definition at line 12 of file svg_generator.h.

5.5.1.4 PLOT_HEIGHT

```
#define PLOT_HEIGHT 300
```

Definition at line 10 of file svg_generator.h.

5.5.1.5 PLOT_WIDTH

```
#define PLOT_WIDTH 1000
```

Definition at line 9 of file svg_generator.h.

5.5.2 Typedef Documentation

5.5.2.1 svg_context

```
typedef struct svg_context svg_context
```

5.5.3 Function Documentation

5.5.3.1 generate_html_file()

```
void generate_html_file (
            FILE * f )
```

Write html + svg plots to the file.

Function for processing data from motors and writing it as html file with svg plots It will display 3 plots: Actual motor position (absolute value) Desire motor position (absolute value) Pwm duty cycle (+- 100%)

The graphs should show at least 2 seconds of history with time resolution 5 ms.

Parameters

desc	: FILE* - file descriptor

Definition at line 22 of file svg_generator.c.

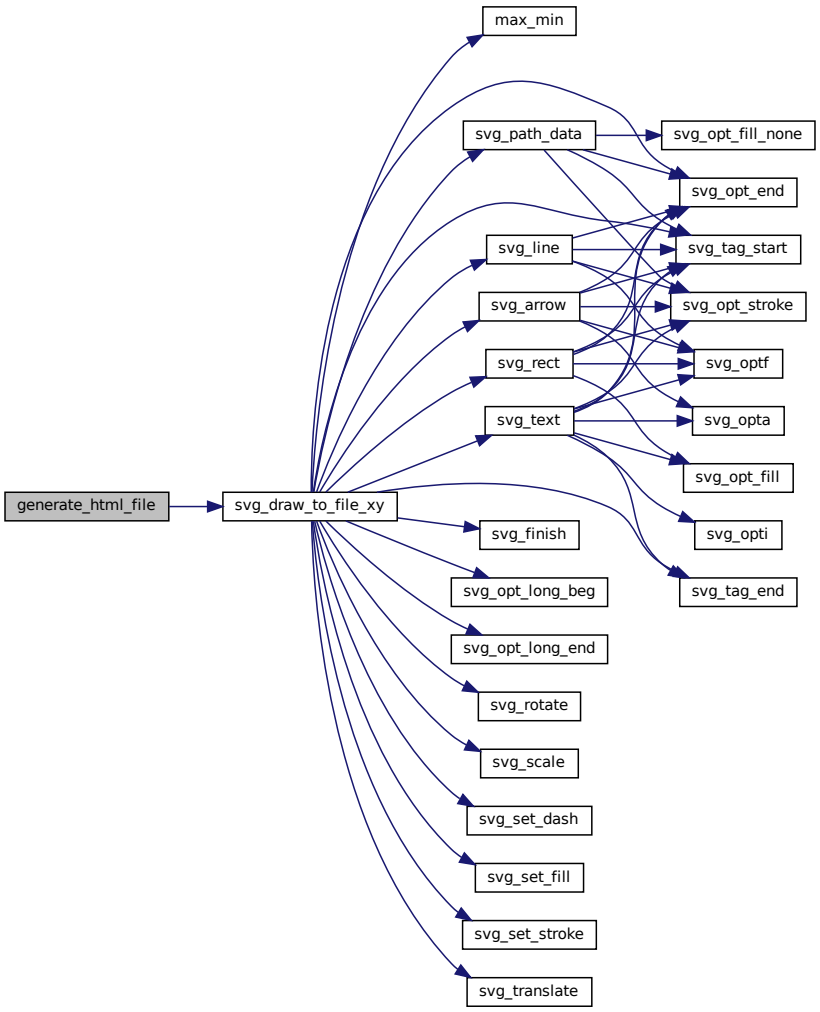
```
22                                  {
23     float x_actual[HISTORY_SIZE];
24     float y_actual[HISTORY_SIZE];
25     float y_desired[HISTORY_SIZE];
26     float y_pwm[HISTORY_SIZE];
27     int i;
28     uint32_t s = motor_history.array_start_index;
29     int copy_array_idx = 0;
30     for (i = s; i != (s + HISTORY_SIZE) % HISTORY_CYCLIC_ARRAY_SIZE; i = (i + 1) %
31                                                                     HISTORY_CYCLIC_ARRAY_SIZE,
       copy_array_idx++) {
32         y_desired[copy_array_idx] = (float) motor_history.desired_position[i];
33         y_actual[copy_array_idx] = (float) motor_history.position[i];
34         y_pwm[copy_array_idx] = (float) motor_history.pwm_duty[i];
35
36         x_actual[copy_array_idx] = (float) motor_history.timestamp[i];
37     }
38
39     // creating svg part
```

```
40     svg_draw_to_file_xy(f, "#ff0000", x_actual, y_actual, HISTORY_SIZE, "t", "Actual position (abs
       units)");
41     svg_draw_to_file_xy(f, "#00ff00", x_actual, y_desired, HISTORY_SIZE, "t", "Desired position (abs
       units)");
42     svg_draw_to_file_xy(f, "#0000ff", x_actual, y_pwm, HISTORY_SIZE, "t", "PWM duty cicle, +-100%");
43 }
```

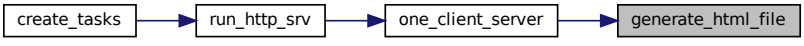
References motor_history_t::array_start_index, motor_history_t::desired_position, HISTORY_CYCLIC_ARRAY_↩ SIZE, HISTORY_SIZE, motor_history, motor_history_t::position, motor_history_t::pwm_duty, svg_draw_to_file_↩ xy(), and motor_history_t::timestamp.

Referenced by one_client_server().

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.3.2 svg_draw_to_file_xy()

```
void svg_draw_to_file_xy (
              FILE * f,
              const char * color,
              float * x,
              float * y,
              size_t count,
              const char * x_label,
              const char * y_label )
```

Plot x-y data to SVG file

Parameters

f	opened file for writing
color	line color
y	data values
x	ordered x values from min to max
count	elements count
x_label	label for Ax
y_label	label for Ay

Definition at line 329 of file svg_generator.c.

```
333                                                                    {
334     float min_y, min_x, max_y, max_x;
335     float height, width, t;
336     float mx, my;
337     float tx, ty;
338     char temp[30];
339 //    svg_context *ctx;
340     if (count == 0)return;
341     min_x = x[0];
342     max_x = x[count - 1];
343     /* Find minimum and maximium of Y axis. X mast be already ordered*/
344     max_min(y, &min_y, &max_y, count);
345     width = max_x - min_x;
346     height = max_y - min_y;
347
348     struct svg_context ctx_obj = {
349             .text_size = 14,
350             .stroke_width = 1,
351             .stroke = 1,
352             .fill = 0,
353             .f = f,
354             .width = PLOT_WIDTH,
355             .height = PLOT_HEIGHT,
356             .padding = 50,
357             .flip = 1,
358             .mx = 1,
359             .my = 1,
360             .ofx = 0,
361             .ofy = 0,
362     };
363     svg_context *ctx = &ctx_obj;
364     strcpy(ctx->fill_color, "#ffffff\0");
365     strcpy(ctx->stroke_color, "#000000\0");
366     strcpy(ctx->text_family, "monospace\0");
367     strcpy(ctx->dash_array, "\0");
368
369     /* Begin document */
370
371     fprintf(ctx->f, "<svg version=\"1.2\" width=\"%i\" height=\"%i\"
      xmlns=\"http://www.w3.org/2000/svg\">",
372             ctx->width, ctx->height);
373     mx = (float) (ctx->width - 2 * ctx->padding) / (float) width;
```

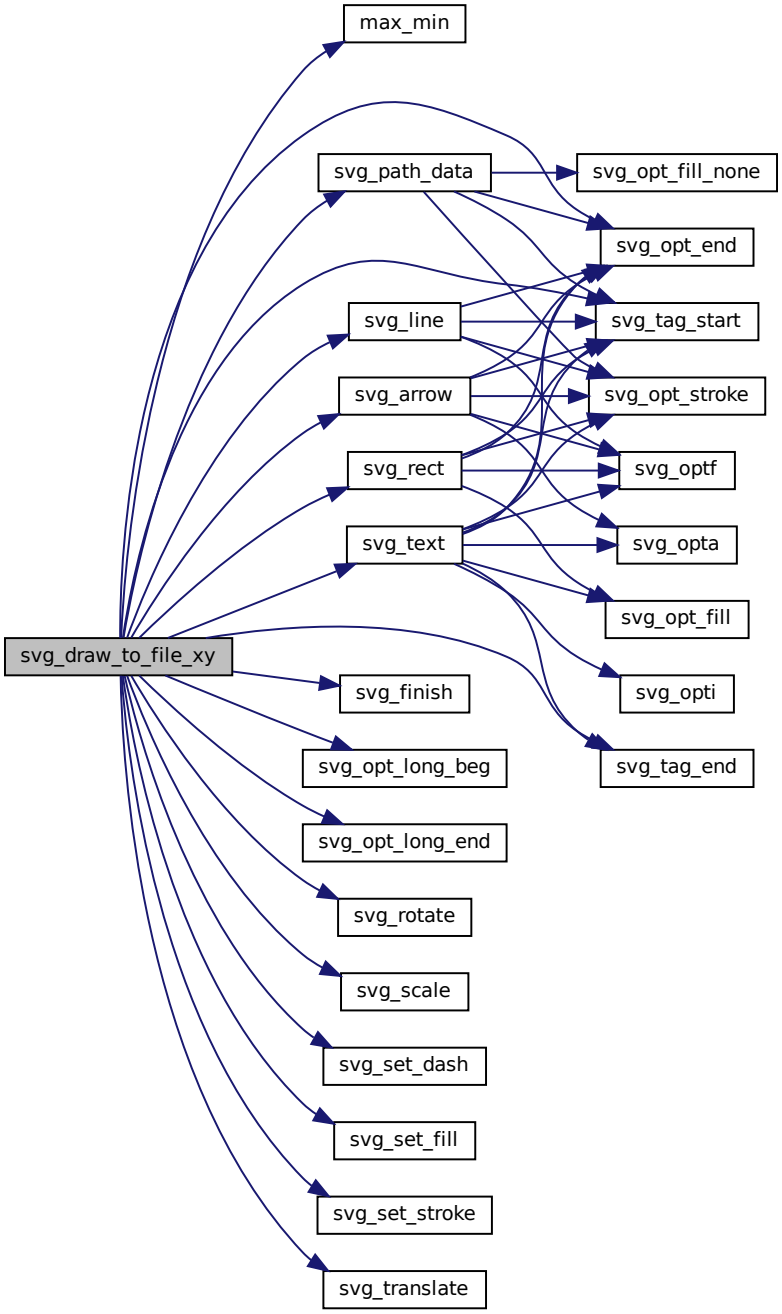
```
374     my = (float) (ctx->height - 2 * ctx->padding) / (float) height;
375     fprintf(f, "<defs>\n");
376     MARKER_ARROW;
377     fprintf(f, "</defs>\n");
378     svg_set_fill(ctx, "#ffffff");
379     svg_rect(ctx, 0, 0, (float) ctx->width, (float) ctx->height);
380
381     /*Vertical lines*/
382     svg_set_stroke(ctx, "#aaaaaa");
383
384     ctx->stroke_width = 1;
385     for (t = min_x + width / 10; t < max_x; t = t + width / 10) {
386         tx = (t - min_x) * mx + (float) ctx->padding;
387
388         svg_set_dash(ctx, "5,5");
389         svg_line(ctx, tx, (float) (ctx->height - ctx->padding), tx, (float) ctx->padding);
390         svg_set_dash(ctx, "");
391         svg_tag_start(ctx, "g", 1);
392         {
393             sprintf(temp, "%i", (int) t);
394             svg_opt_long_beg(ctx, "transform");
395             svg_rotate(ctx, -90, (int) tx - 3, ctx->height - ctx->padding - 10);
396             svg_translate(ctx, (int) tx - 3, ctx->height - ctx->padding - 10);
397             svg_opt_long_end(ctx);
398             svg_opt_end(ctx, 0);
399
400             svg_text(ctx, temp, 0, 0);
401         }
402         svg_tag_end(ctx, "g");
403     }
404     /*Horizontal lines*/
405     svg_set_stroke(ctx, "#aaaaaa");
406
407     ctx->stroke_width = 1;
408     for (t = min_y + height / 10; t < max_y; t = t + height / 10.0) {
409         ty = (float) ctx->height - (t - min_y) * my - (float) ctx->padding;
410         svg_set_dash(ctx, "5,5");
411         svg_line(ctx, (float) ctx->padding, ty, (float) (ctx->width - ctx->padding), ty);
412         svg_set_dash(ctx, "");
413         sprintf(temp, "%i", (int) t);
414         svg_text(ctx, temp, (float) ctx->padding + 3, ty - 3);
415     }
416
417     svg_tag_start(ctx, "g", 1);
418     {
419         ctx->mx = mx;
420         ctx->my = my;
421         ctx->ofx = min_x;
422         ctx->ofy = min_y;
423         svg_opt_long_beg(ctx, "transform");
424         svg_translate(ctx, ctx->padding, ctx->padding);
425         svg_scale(ctx, 1, -1);
426         svg_translate(ctx, 0, -ctx->height + 2 * ctx->padding);
427         svg_opt_long_end(ctx);
428         svg_opt_end(ctx, 0);
429
430
431         /* Plot data */
432         svg_set_stroke(ctx, color);
433         svg_set_dash(ctx, "");
434         svg_path_data(ctx, x, y, count);
435         svg_set_stroke(ctx, "#000000");
436
437     }
438     svg_tag_end(ctx, "g");
439     ctx->mx = 1;
440     ctx->my = 1;
441     ctx->ofx = 0;
442     ctx->ofy = 0;
443     sprintf(temp, "%i", (int) max_y);
444     svg_text(ctx, temp, (float) ctx->padding + 3, (float) ctx->padding);
445     sprintf(temp, "%i", (int) max_x);
446     svg_tag_start(ctx, "g", 1);
447     {
448         svg_opt_long_beg(ctx, "transform");
449         svg_rotate(ctx, -90,
450                    (int) (ctx->width - ctx->padding - ctx->text_size / 2),
451                    (int) (ctx->height - ctx->padding - ctx->text_size));
452         svg_translate(ctx,
453                       (int) (ctx->width - ctx->padding - ctx->text_size),
454                       (int) (ctx->height - ctx->padding - ctx->text_size));
455         svg_opt_long_end(ctx);
456         svg_opt_end(ctx, 0);
457         svg_text(ctx, temp, 0, 0);
458     }
459     svg_tag_end(ctx, "g");
460     /*axis*/
```

```
461     svg_set_fill(ctx, "#000000");
462     svg_set_stroke(ctx, "#000000");
463     svg_set_dash(ctx, "");
464     ctx->stroke_width = 2;
465     ctx->text_size = (ctx->padding) / 2;
466 
467     /* Y */
468     svg_set_stroke(ctx, "#000000");
469     svg_set_dash(ctx, "");
470 
471     svg_arrow(ctx,
472               (float) (ctx->padding),
473               (float) (ctx->height - ctx->padding / 2.0),
474               (float) (ctx->padding),
475               (float) (ctx->padding / 2.0));
476     ctx->stroke_width = 1;
477     svg_text(ctx, y_label,
478              (float) (ctx->padding + ctx->text_size / 4.0),
479              (float) (ctx->padding / 2.0 + ctx->text_size / 4.0));
480 
481     sprintf(temp, "%i", (int) x[0]);
482 
483     ctx->text_size = 14;
484     svg_tag_start(ctx, "g", 1);
485     {
486         svg_opt_long_beg(ctx, "transform");
487         svg_rotate(ctx, -90, (int) (ctx->padding - ctx->text_size / 2),
488                    (int) (ctx->height - ctx->padding - ctx->text_size));
489         svg_translate(ctx, (int) (ctx->padding - ctx->text_size),
490                       (int) (ctx->height - ctx->padding - ctx->text_size));
491         svg_opt_long_end(ctx);
492         svg_opt_end(ctx, 0);
493         svg_text(ctx, temp, 0, 0);
494     }
495     svg_tag_end(ctx, "g");
496     ctx->text_size = ctx->padding / 2;
497 
498     ctx->stroke_width = 2;
499 
500     /* X */
501     svg_arrow(ctx,
502               (float) ctx->padding / 2,
503               (float) (ctx->height - ctx->padding),
504               (float) (ctx->width - ctx->padding / 2.0),
505               (float) (ctx->height - ctx->padding));
506     ctx->stroke_width = 1;
507     ctx->text_size = ctx->padding / 2;
508 
509     svg_text(ctx, x_label,
510              (float) (ctx->width - strlen(x_label) * ctx->text_size),
511              (float) (ctx->height - ctx->padding - ctx->text_size / 2.0));
512     sprintf(temp, "%i", (int) min_y);
513     ctx->text_size = 14;
514     svg_text(ctx, temp, (float) ctx->padding + 3, (float) (ctx->height - ctx->padding +
    ctx->text_size));
515 
516     svg_finish(ctx);
517 }
```

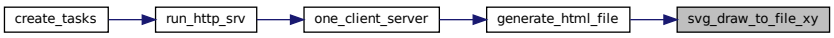
References svg_context::dash_array, svg_context::f, svg_context::fill_color, svg_context::height, MARKER_AR↩ ROW, max_min(), svg_context::mx, svg_context::my, svg_context::ofx, svg_context::ofy, svg_context::padding, PLOT_HEIGHT, PLOT_WIDTH, svg_context::stroke_color, svg_context::stroke_width, svg_arrow(), svg_finish(), svg_line(), svg_opt_end(), svg_opt_long_beg(), svg_opt_long_end(), svg_path_data(), svg_rect(), svg_rotate(), svg_scale(), svg_set_dash(), svg_set_fill(), svg_set_stroke(), svg_tag_end(), svg_tag_start(), svg_text(), svg_↩ translate(), svg_context::text_family, svg_context::text_size, and svg_context::width.

Referenced by generate_html_file().

Here is the call graph for this function:



Here is the caller graph for this function:

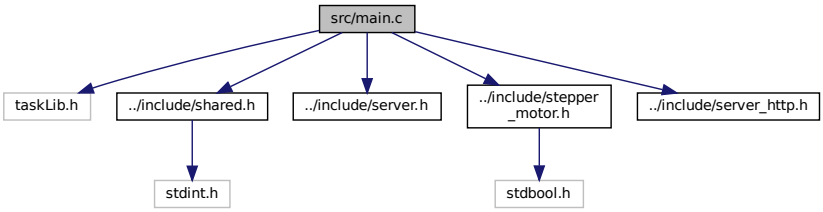


5.6 README.md File Reference

5.7 src/main.c File Reference

```
#include <taskLib.h>
#include "../include/shared.h"
#include "../include/server.h"
#include "../include/stepper_motor.h"
#include "../include/server_http.h"
```

Include dependency graph for main.c:



Functions

- void create_tasks (bool transmiter, char ∗ip_address)

  Main function of our project Create tasks - function for tasks creation, setting all parameters and spawning. Main entry point in a whole project for both modes.

5.7.1 Function Documentation

5.7.1.1 create_tasks()

```
void create_tasks (
            bool transmiter,
            char * ip_address )
```

Main function of our project Create tasks - function for tasks creation, setting all parameters and spawning. Main entry point in a whole project for both modes.

Parameters

transmiter	- bool - flag for differentiation between two modes - receiver and transmitter
ip_address	- char∗ - ip address of the UDP receiver or NULL if the board is receiver itself

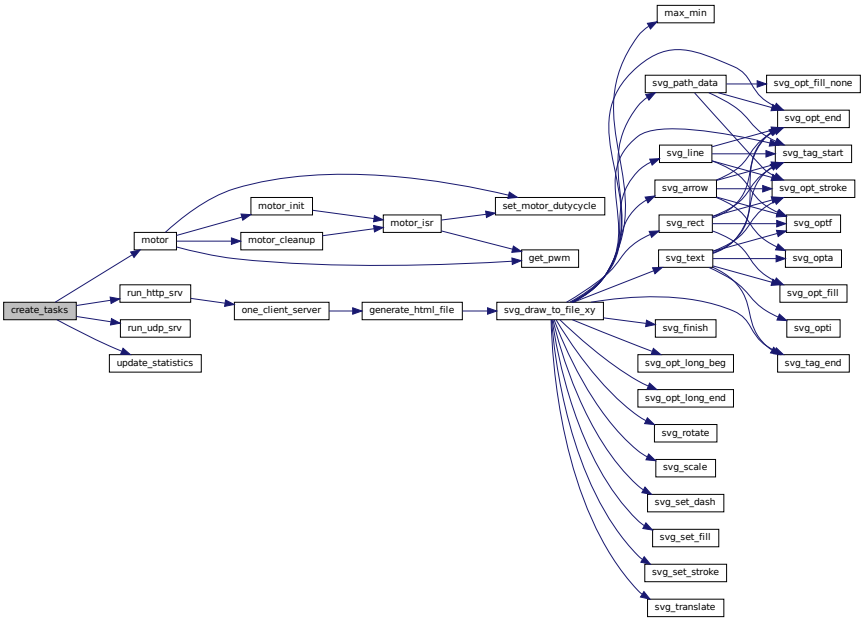
Definition at line 15 of file main.c.

```
15                                          {
```

```
16      sysClkRateSet(CLOCK_RATE);
17      sysTimestampEnable();
18      motor_history.timestamp_start = sysTimestamp();
19
20      printf("%lu\n", sizeof(motor_history.desired_position));
21      // setting all arrays to zero, because we are reading from them
22      memset(motor_history.desired_position, 0, sizeof(motor_history.desired_position));
23      memset(motor_history.position, 0, sizeof(motor_history.position));
24      memset(motor_history.pwm_duty, 0, sizeof(motor_history.pwm_duty));
25
26      taskSpawn("tMotor_driver", MOTOR_DRIVER_PRIOROTY, 0, 4096, (FUNCPTR)motor, !transmiter, 0, 0, 0, 0,
        0, 0, 0, 0, 0);
27      taskSpawn("tUDP_server", UDP_COMMUNICATION_PRIORITY, 0, 4096, (FUNCPTR)run_udp_srv,
        (_Vx_usr_arg_t)ip_address, DEFAULT_PORT, 0, 0, 0, 0, 0, 0, 0, 0);
28
29      if (!transmiter) {
30          taskSpawn("tHTTP_server", HTTP_SERVER_PRIORITY, 0, 16384 * 4, (FUNCPTR)run_http_srv, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0);
31          taskSpawn("tStatistics_monitor", STATISTICS_MONITOR_PRIORITY, 0, 4096,
        (FUNCPTR)update_statistics, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
32      }
33  }
```

References CLOCK_RATE, DEFAULT_PORT, motor_history_t::desired_position, HTTP_SERVER_PRIORITY, motor(), MOTOR_DRIVER_PRIOROTY, motor_history, motor_history_t::position, motor_history_t::pwm_duty, run_↩ http_srv(), run_udp_srv(), STATISTICS_MONITOR_PRIORITY, motor_history_t::timestamp_start, UDP_COMM↩ UNICATION_PRIORITY, and update_statistics().

Here is the call graph for this function:

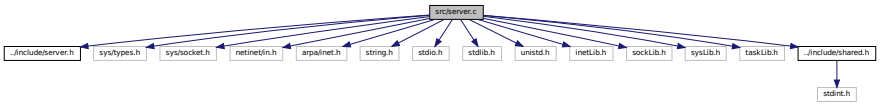


5.8 src/server.c File Reference

```
#include "../include/server.h"
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <unistd.h>
#include <inetLib.h>
#include <sockLib.h>
#include <sysLib.h>
#include <taskLib.h>
#include "../include/shared.h"
```

Include dependency graph for server.c:



Functions

- void run_udp_srv (char *ip_address, int port)

  Run udp server for sharing data We use this function for communication between two microzed boards It sends desire position from one board to another using BSD sockets.

5.8.1 Function Documentation

5.8.1.1 run_udp_srv()

```
void run_udp_srv (
            char * ip_address,
            int port )
```

Run udp server for sharing data We use this function for communication between two microzed boards It sends desire position from one board to another using BSD sockets.

Parameters

ip_address	: char* - server's ip address
port	: int - udp port for sockets

Definition at line 25 of file server.c.

```
25                                          {
26     // init semaphore
27     SEM_ID sem_motor_pwm_change = semOpen("/sem_motor_pwm_change", SEM_TYPE_COUNTING, SEM_EMPTY,
       SEM_Q_FIFO, OM_CREATE, NULL);
28
29     // information_preprocessing
30     if (sysTimestampEnable() == ERROR) {
31         perror("Time stamps could not be enabled");
32         return;
33     }
34
35     int sockd;
36     struct sockaddr_in my_name, cli_name, srv_addr;
37     socklen_t addrlen;
38     const int buf_len = sizeof(int32_t);
```

```
39      int i, status;
40
41      // Common part for both receiver and the transmitter
42      sockd = socket(AF_INET, SOCK_DGRAM, 0);
43      if (sockd == -1) {
44          perror("Could not open the socket...");
45          exit(1);
46      }
47
48      my_name.sin_family = AF_INET;
49      my_name.sin_addr.s_addr = INADDR_ANY;
50      my_name.sin_port = htons(port);
51
52      if (bind(sockd, (struct sockaddr *) &my_name, sizeof(my_name)) == -1) {
53          perror("Could not bind the socket...");
54          close(sockd);
55          exit(2);
56      }
57
58      addrlen = sizeof(cli_name);
59
60
61      if (ip_address == NULL) {
62          // Running as a receiver
63          //      set int32_t motor_driver_shared.desired_position
64  #ifdef DEBUG_OUTPUT
65          printf("Runnign as a receiver...\n");
66  #endif
67
68          while (1) {
69              status = recvfrom(sockd, &(motor_driver_shared.desired_position), buf_len, 0, (struct
      sockaddr *) &cli_name, & addrlen);
70              semGive(sem_motor_pwm_change);
71  #ifdef DEBUG_OUTPUT
72              printf("received %i\n", motor_driver_shared.desired_position);
73  #endif
74          }
75          close(sockd);
76      } else {
77  #ifdef DEBUG_OUTPUT
78          printf("Runnign as a transmitter...\n");
79  #endif
80          // Running as a transmitter
81          //      transfer int32_t motor_driver_shared.desired_position
82          srv_addr.sin_family = AF_INET;
83          inet_aton(ip_address, &srv_addr.sin_addr);
84          srv_addr.sin_port = htons(port);
85
86          i = 0;
87          while (1) {
88              sendto(sockd, &(motor_driver_shared.position), buf_len, 0, (struct sockaddr *) &srv_addr,
      sizeof(srv_addr));
89  #ifdef DEBUG_OUTPUT
90              printf("sent %i\n", motor_driver_shared.position);
91  #endif
92              taskDelay(SERVER_SEND_DELAY);
93          }
94          close(sockd);
95      }
96      printf("Measurement finished\n");
97  }
```

References motor_driver_shared_t::desired_position, motor_driver_shared, motor_driver_shared_t::position, sem_motor_pwm_change, and SERVER_SEND_DELAY.

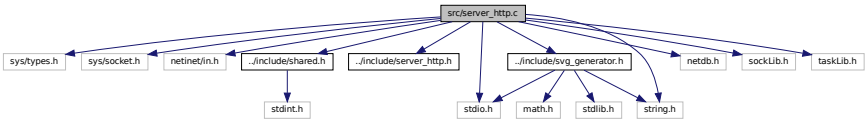
Referenced by create_tasks().

Here is the caller graph for this function:

5.9 src/server_http.c File Reference

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include "../include/shared.h"
#include "../include/server_http.h"
#include "../include/svg_generator.h"
#include <netdb.h>
#include <stdio.h>
#include <sockLib.h>
#include <string.h>
#include <taskLib.h>
```

Include dependency graph for server_http.c:



Functions

- static void one_client_server (int client_fd)
- void run_http_srv ()

5.9.1 Function Documentation

5.9.1.1 one_client_server()

```
static void one_client_server (
            int client_fd )  [static]
```

Entry point of the task that serves one request of one client of the server Function sends an HTML file with plots if receives a GET request to the root

Parameters

client↩ _fd	Open file descriptor for communication with the client through TCP socker

Definition at line 19 of file server_http.c.

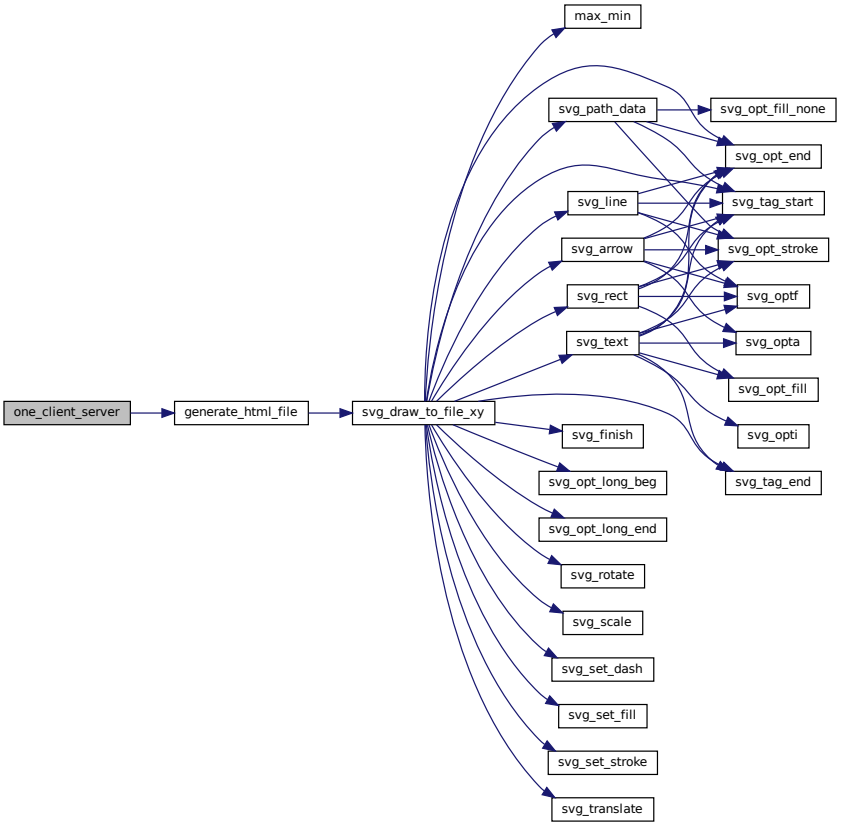
```
19                                          {
20     char buf[HTTP_RECCEIVER_BUF_SIZE];
21     int message_size;
22     if ((message_size = recv(client_fd, buf, sizeof(buf) - 1, 0))) {
23         if (message_size == -1) {
24             perror("Receiving message through the socket");
25             close(client_fd);
```

```
26              return;
27          }
28          buf[message_size] = '\0';
29          // Ignoring all headers for now. TODO:
30          // Dealing only with GET request to the root
31 #ifdef DEBUG_OUTPUT
32          int i;
33          for (i = 0; i < message_size; i++)
34              printf("%c", buf[i]);
35
36 #endif
37         // Firstly, check if the request is the GET one to the root
38          if (message_size > 5) {
39              if (strncmp(buf, "GET / ", 6) == 0) {
40                  printf("Sending SVG..\n");
41                  FILE *f = fdopen(client_fd, "w");i
42
43                  // Send the only header
44                  fprintf(f, "HTTP/1.0 200 OK\r\n\r\n");
45
46                  // Send html first tags with JS function for reloading page
47                  fprintf(f, "<!DOCTYPE html>\n<html>\n");
48                  fprintf(f, "<body onload=\"setTimeout(function(){location.reload()}, 400);\">");
49
50                  // Generate plots in the file
51                  generate_html_file(f);
52
53                  // Send html closing tags
54                  fprintf(f, "</body>\n</html>");
55                  fclose(f);
56 #ifdef DEBUG_OUTPUT
57                  printf("Image sent\n");
58 #endif
59              }
60          }
61      }
62
63      close(client_fd);
64 }
```

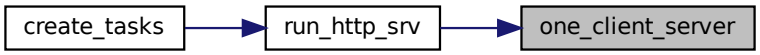
References generate_html_file(), and HTTP_RECCEIVER_BUF_SIZE.

Referenced by run_http_srv().

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.1.2 run_http_srv()

```
void run_http_srv ( )
```

Entry point of the task for running the HTTP server The task creates a new task for serving each request from any client

Definition at line 70 of file server_http.c.

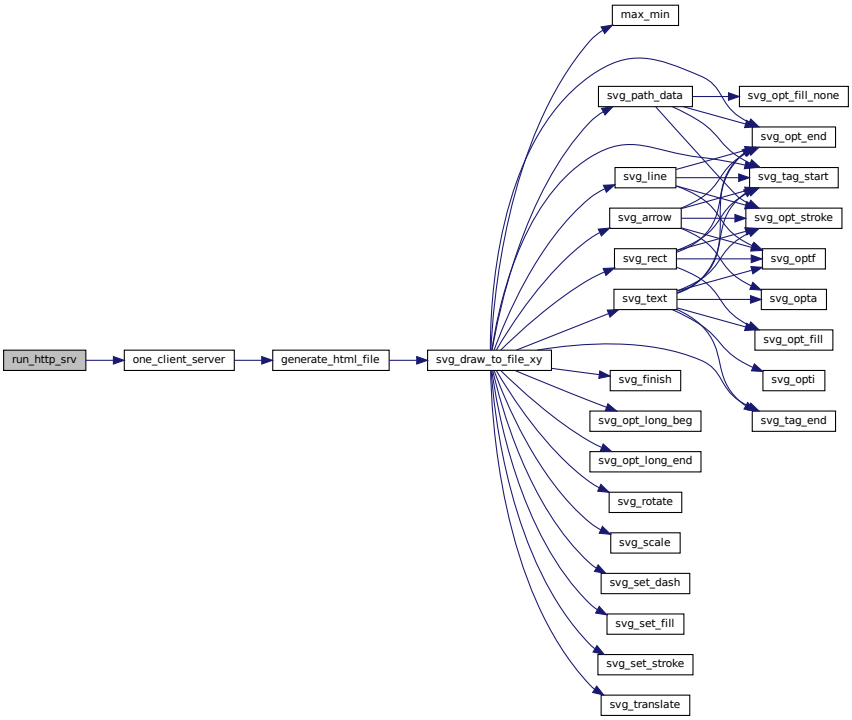
```
70                      {
```

```
71      int s;
72      int new_fd;
73      struct sockaddr_in serverAddr;
74      struct sockaddr_in clientAddr;
75      socklen_t sockAddrSize;
76
77      sockAddrSize = sizeof(struct sockaddr_in);
78      bzero((char *) &serverAddr, sizeof(struct sockaddr_in));
79      serverAddr.sin_family = AF_INET;
80      serverAddr.sin_port = htons(HTTP_SERVER_PORT);
81      serverAddr.sin_addr.s_addr = htonl(INADDR_ANY);
82
83      s = socket(AF_INET, SOCK_STREAM, 0);
84      if (s < 0) {
85          printf("Error: www: socket(%d)\n", s);
86          return;
87      }
88
89
90      if (bind(s, (struct sockaddr *) &serverAddr, sockAddrSize) == ERROR) {
91          printf("Error: www: bind\n");
92          return;
93      }
94
95      if (listen(s, HTTP_SERVER_MAX_CONNECTIONS) == ERROR) {
96          perror("www listen");
97          close(s);
98          return;
99      }
100
101 #ifdef DEBUG_OUTPUT
102     printf("www server running\n");
103 #endif
104
105     char task_name_buf[100];
106
107     int client_num = 0;
108     while (1) {
109         /* accept waits for somebody to connect and the returns a new file descriptor */
110         if ((new_fd = accept(s, (struct sockaddr *) &clientAddr, &sockAddrSize)) == ERROR) {
111             perror("www accept");
112             close(s);
113             return;
114         }
115         sprintf(task_name_buf, "tOneClientServer%d", client_num++);
116         if (taskSpawn(task_name_buf, HTTP_SERVER_PRIORITY, VX_FP_TASK, 65536, (FUNCPTR)
    one_client_server, new_fd, 0, 0, 0, 0, 0, 0, 0, 0) == ERROR) {
117             perror("Creating task for communication with an client");
118             close(s);
119             close(new_fd);
120             return;
121         }
122     }
123 }
```

References HTTP_SERVER_MAX_CONNECTIONS, HTTP_SERVER_PORT, HTTP_SERVER_PRIORITY, and one_client_server().

Referenced by create_tasks().

Here is the call graph for this function:

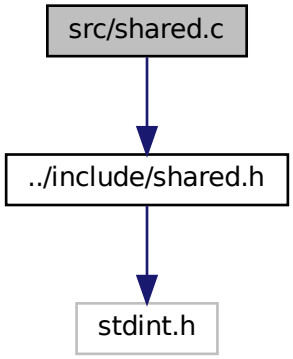


Here is the caller graph for this function:

5.10 src/shared.c File Reference

```
#include "../include/shared.h"
```

Include dependency graph for shared.c:



Variables

- motor_driver_shared_t motor_driver_shared
- motor_history_t motor_history

5.10.1 Variable Documentation

5.10.1.1 motor_driver_shared

```
motor_driver_shared_t motor_driver_shared
```

Initial value:

```
= {
        .desired_position = 0,
        .irq_count = 0,
        .position = 0,
        .pwm_duty = 0
}
```

Definition at line 3 of file shared.c.

Referenced by motor(), motor_init(), motor_isr(), run_udp_srv(), and update_statistics().

5.10.1.2 motor_history

```
motor_history_t motor_history
```

Initial value:

```
= {
        .array_start_index = 0
}
```

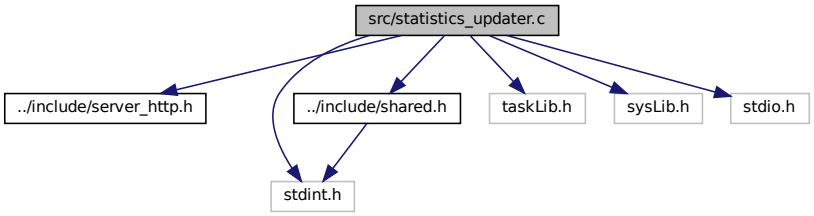
Definition at line 10 of file shared.c.

Referenced by create_tasks(), generate_html_file(), and update_statistics().

5.11 src/statistics_updater.c File Reference

```
#include "../include/server_http.h"
#include <stdint.h>
#include "../include/shared.h"
#include <taskLib.h>
#include <sysLib.h>
#include <stdio.h>
```

Include dependency graph for statistics_updater.c:



Functions

- void update_statistics ()

5.11.1 Function Documentation

5.11.1.1 update_statistics()

```
void update_statistics ( )
```

Entry point of task that contonuously updates motor history statistics

Written using Ostrich algorithm - we ignore data race in this function Obviously it can happen (somewhere on the bounds) and we realise it But to use mutex/semaphore here will be a huge overhead for the whole system.

Definition at line 15 of file statistics_updater.c.

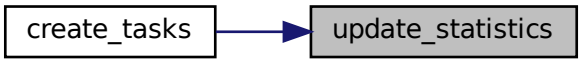
```
15                          {
16     uint32_t prev_timestamp = sysTimestamp(), cur_timestamp;
17     uint64_t timestamp64 = 0;
18
19     while (1) {
20         cur_timestamp = sysTimestamp();
21         timestamp64 += cur_timestamp - prev_timestamp;
22         prev_timestamp = cur_timestamp;
23
24         uint32_t i = (motor_history.array_start_index + HISTORY_SIZE) % HISTORY_CYCLIC_ARRAY_SIZE;
25         motor_history.desired_position[i] = motor_driver_shared.desired_position;
26         motor_history.position[i] = motor_driver_shared.position;
27         motor_history.pwm_duty[i] = motor_driver_shared.pwm_duty;
28         motor_history.timestamp[i] = (timestamp64 * 1000) / sysTimestampFreq();
29
30         motor_history.array_start_index = (motor_history.array_start_index + 1) %
       HISTORY_CYCLIC_ARRAY_SIZE;
31         taskDelay(3);
```

```
32     }
33 }
```

References motor_history_t::array_start_index, motor_driver_shared_t::desired_position, motor_history_t::desired_position, HISTORY_CYCLIC_ARRAY_SIZE, HISTORY_SIZE, motor_driver_shared, motor_history, motor_driver_shared_t::position, motor_history_t::position, motor_driver_shared_t::pwm_duty, motor_history_t::pwm_duty, and motor_history_t::timestamp.

Referenced by create_tasks().

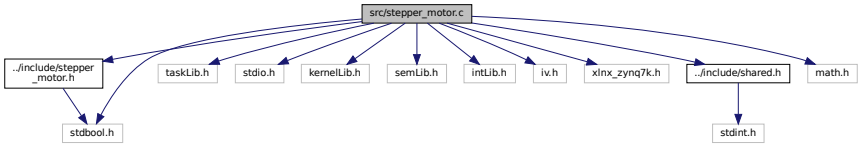
Here is the caller graph for this function:



5.12 src/stepper_motor.c File Reference

```
#include "../include/stepper_motor.h"
#include <taskLib.h>
#include <stdio.h>
#include <kernelLib.h>
#include <semLib.h>
#include <intLib.h>
#include <iv.h>
#include <xlnx_zynq7k.h>
#include <stdbool.h>
#include "../include/shared.h"
#include <math.h>
```

Include dependency graph for stepper_motor.c:



Macros

- #define TRANSITION(old, new) (((old) << 2) | new)

Functions

- static void set_motor_dutycycle (int8_t duty_cycle_percent)

  Set duty cycle in percentage Set duty cycle for controlling the motor to the values from -100 to 100, where -100 means max speed backwards, 100 means max speed forward, 0 means no movement and so on.
- static int8_t get_pwm (int32_t distance)

  Get new PWM value from the distance between desired and real position.
- void motor_isr (void)

  Motor interrupt handler Interrupt handler for handling intrrupts on the stepper motor phase change.
- void motor_init (void)

  Initialize the step motor.
- void motor_cleanup (void)

  interrupt handler and registers
- void motor (bool set_desired)

  Entry point to the DKM.

Variables

- static const int TRANSITION_TABLE [16]
- static SEM_ID sem_motor_pwm_change
- static uint8_t motor_state
- static uint8_t prev_motor_state
- static bool set_position_to_desired

5.12.1 Macro Definition Documentation

5.12.1.1 TRANSITION

```
#define TRANSITION(
              old,
              new ) (((old) << 2) | new)
```

Assigns transition from old state to the new one an index

Parameters

old	Old motor state
new	New motor state

Definition at line 18 of file stepper_motor.c.

5.12.2 Function Documentation

5.12.2.1 get_pwm()

```
static int8_t get_pwm (
            int32_t distance )  [inline], [static]
```

Get new PWM value from the distance between desired and real position.

Parameters

distance	distance in steps between the desired and real position

Returns

new PWM value in range (-100, 100)

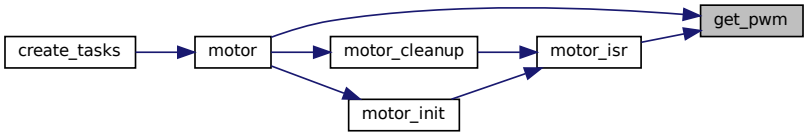
Definition at line 58 of file stepper_motor.c.

```
58                                              {
59     distance = max(min(distance, 200), -200);
60     int8_t pwm = min(max(((distance * abs(distance)) » 8), -100), 100);
61     if (distance < 0 && pwm > -MIN_PWM_DUTY) {
62         return -MIN_PWM_DUTY;
63     } else if (distance > 0 && pwm < MIN_PWM_DUTY) {
64         return MIN_PWM_DUTY;
65     } else {
66         return pwm;
67     }
68 }
```

References MIN_PWM_DUTY.

Referenced by motor(), and motor_isr().

Here is the caller graph for this function:



5.12.2.2 motor()

```
void motor (
            bool set_desired )
```

Entry point to the DKM.

Parameters

set_desired	: true if the driver should set position of the motor and not just monitor values

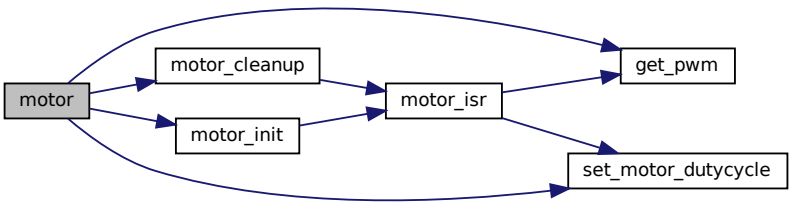
Definition at line 134 of file stepper_motor.c.

```
134                                {
135     set_position_to_desired = set_desired;
136     // Init semaphore for changing PWM duty requests
137     sem_motor_pwm_change = semOpen("/sem_motor_pwm_change", SEM_TYPE_COUNTING, SEM_EMPTY, SEM_Q_FIFO,
        OM_CREATE, NULL);
138
139     motor_init();
140
141     while (1) {
142         // Wait for the request to recalculate the duty cycle
143         semTake(sem_motor_pwm_change, WAIT_FOREVER);
144         if (set_desired) {
145             // Disable interrupts to prevent data race (when interrupt handler sets duty cycle too)
146             int8_t duty = get_pwm(motor_driver_shared.desired_position - motor_driver_shared.position);
147             intDisable(INT_LVL_GPIO);
148             set_motor_dutycycle(duty);
149             motor_driver_shared.pwm_duty = duty;
150             intEnable(INT_LVL_GPIO);
151         }
152     }
153
154     semClose(sem_motor_pwm_change);
155     motor_cleanup();
156 }
```

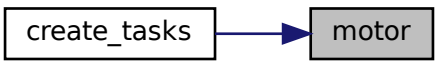
References motor_driver_shared_t::desired_position, get_pwm(), motor_cleanup(), motor_driver_shared, motor_↩ init(), motor_driver_shared_t::position, motor_driver_shared_t::pwm_duty, sem_motor_pwm_change, set_motor↩ _dutycycle(), and set_position_to_desired.

Referenced by create_tasks().

Here is the call graph for this function:



Here is the caller graph for this function:



5.12.2.3 motor_cleanup()

```
void motor_cleanup (
            void  )
```

interrupt handler and registers

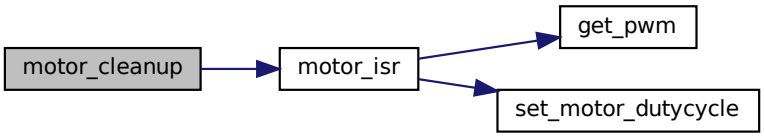
Definition at line 124 of file stepper_motor.c.

```
124                         {
125     GPIO_INT_DISABLE = MOTOR_IRQ_PIN;
126     intDisable(INT_LVL_GPIO);
127     intDisconnect(INUM_TO_IVEC(INT_LVL_GPIO), motor_isr, 0);
128 }
```

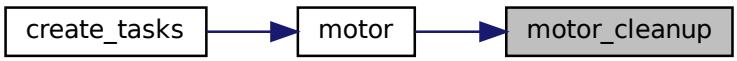
References GPIO_INT_DISABLE, MOTOR_IRQ_PIN, and motor_isr().

Referenced by motor().

Here is the call graph for this function:



Here is the caller graph for this function:



5.12.2.4 motor_init()

```
void motor_init (
            void  )
```

Initialize the step motor.

Definition at line 96 of file stepper_motor.c.

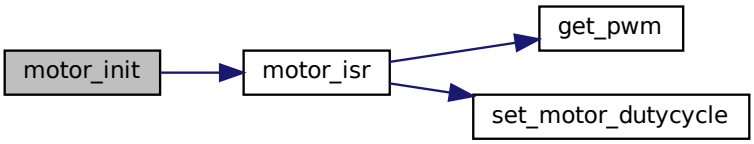
```
96                      {
97     motor_driver_shared.position = 0;
98     motor_driver_shared.irq_count = 0;
99
```

```
100     GPIO_INT_STATUS = MOTOR_IRQ_PIN; /* reset status */
101     GPIO_DIRM = 0x0;                 /* set as input */
102     GPIO_INT_TYPE = MOTOR_IRQ_PIN;   /* interrupt on edge */
103     GPIO_INT_POLARITY = 0x0;         /* falling edge */
104     GPIO_INT_ANY = 0x0;              /* ignore rising edge */
105     GPIO_INT_ENABLE = MOTOR_IRQ_PIN; /* enable interrupt on MOTOR_IRQ pin */
106
107     PWM_PERIOD_REGISTER = PWM_PERIOD; // Set PWM period to 20 kHz
108     MOTOR_CR = (1 « MOTOR_CR_PWM_ENABLE); // Enable PWM by default
109     PWM_DUTY_CR = 0; // No PWM for now
110
111     bool irc_a = (MOTOR_SR & BIT(MOTOR_SR_IRC_A_MON)) != 0;
112     bool irc_b = (MOTOR_SR & BIT(MOTOR_SR_IRC_B_MON)) != 0;
113
114     // TODO: check if this is ok with bools here
115     motor_state = irc_a | (irc_b « 1);
116
117     intConnect(INUM_TO_IVEC(INT_LVL_GPIO), motor_isr, 0);
118     intEnable(INT_LVL_GPIO);         /* enable all GPIO interrupts */
119 }
```

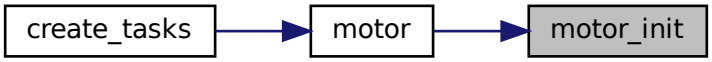
References BIT, GPIO_DIRM, GPIO_INT_ANY, GPIO_INT_ENABLE, GPIO_INT_POLARITY, GPIO_INT_STA↩ TUS, GPIO_INT_TYPE, motor_driver_shared_t::irq_count, MOTOR_CR, MOTOR_CR_PWM_ENABLE, motor_↩ driver_shared, MOTOR_IRQ_PIN, motor_isr(), MOTOR_SR, MOTOR_SR_IRC_A_MON, MOTOR_SR_IRC_B_↩ MON, motor_state, motor_driver_shared_t::position, PWM_DUTY_CR, PWM_PERIOD, and PWM_PERIOD_RE↩ GISTER.

Referenced by motor().

Here is the call graph for this function:



Here is the caller graph for this function:



5.12.2.5 motor_isr()

```
void motor_isr (
            void  )
```

Motor interrupt handler Interrupt handler for handling intrrupts on the stepper motor phase change.

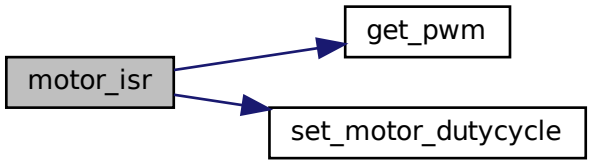
Definition at line 74 of file stepper_motor.c.

```
74                      {
75     uint8_t irc_a = (MOTOR_SR & BIT(MOTOR_SR_IRC_A_MON)) ? 1 : 0;
76     uint8_t irc_b = (MOTOR_SR & BIT(MOTOR_SR_IRC_B_MON)) ? 2 : 0;
77     int8_t duty;
78 
79     prev_motor_state = motor_state;
80     motor_state = irc_a | irc_b;
81     motor_driver_shared.position += TRANSITION_TABLE[TRANSITION(prev_motor_state, motor_state)];
82     motor_driver_shared.irq_count++;
83 
84     if (set_position_to_desired) {
85         duty = get_pwm(motor_driver_shared.desired_position - motor_driver_shared.position);
86         set_motor_dutycycle(duty);
87         motor_driver_shared.pwm_duty = duty;
88     }
89 
90     GPIO_INT_STATUS = MOTOR_IRQ_PIN; /* clear the interrupt */
91 }
```

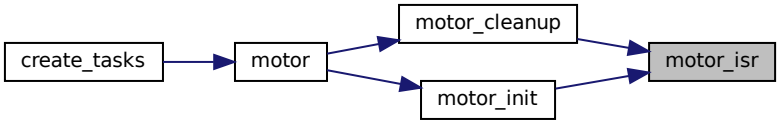
References BIT, motor_driver_shared_t::desired_position, get_pwm(), GPIO_INT_STATUS, motor_driver_shared_t::irq_count, motor_driver_shared, MOTOR_IRQ_PIN, MOTOR_SR, MOTOR_SR_IRC_A_MON, MOTOR_SR_IRC_B_MON, motor_state, motor_driver_shared_t::position, prev_motor_state, motor_driver_shared_t::pwm_duty, set_motor_dutycycle(), set_position_to_desired, TRANSITION, and TRANSITION_TABLE.

Referenced by motor_cleanup(), and motor_init().

Here is the call graph for this function:



Here is the caller graph for this function:



5.12.2.6 set_motor_dutycycle()

```
static void set_motor_dutycycle (
             int8_t duty_cycle_percent )  [inline], [static]
```

Set duty cycle in percentage Set duty cycle for controlling the motor to the values from -100 to 100, where -100 means max speed backwards, 100 means max speed forward, 0 means no movement and so on.

Parameters

duty_cycle_percent	: desired percentage of duty cycle (from -100 to 100)

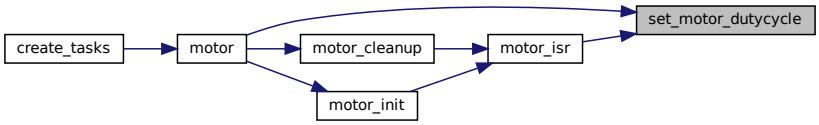
Definition at line 42 of file stepper_motor.c.

```
42                                                          {
43     if (duty_cycle_percent < -100 || duty_cycle_percent > 100) {
44         return;
45     }
46     if (duty_cycle_percent < 0) {
47         PWM_DUTY_CR = ((1 « DUTY_DIR_R) | (((PWM_PERIOD * -duty_cycle_percent) / 100) « DUTY_CYCLE));
48     } else {
49         PWM_DUTY_CR = ((1 « DUTY_DIR_F) | (((PWM_PERIOD * duty_cycle_percent) / 100) « DUTY_CYCLE));
50     }
51 }
```

References DUTY_CYCLE, DUTY_DIR_F, DUTY_DIR_R, PWM_DUTY_CR, and PWM_PERIOD.

Referenced by motor(), and motor_isr().

Here is the caller graph for this function:



5.12.3 Variable Documentation

5.12.3.1 motor_state

```
uint8_t motor_state  [static]
```

Definition at line 33 of file stepper_motor.c.

Referenced by motor_init(), and motor_isr().

5.12.3.2 prev_motor_state

```
uint8_t prev_motor_state  [static]
```

Definition at line 33 of file stepper_motor.c.

Referenced by motor_isr().

5.12.3.3 sem_motor_pwm_change

```
SEM_ID sem_motor_pwm_change  [static]
```

Definition at line 32 of file stepper_motor.c.

Referenced by motor(), and run_udp_srv().

5.12.3.4 set_position_to_desired

```
bool set_position_to_desired  [static]
```

Definition at line 34 of file stepper_motor.c.

Referenced by motor(), and motor_isr().

5.12.3.5 TRANSITION_TABLE

```
const int TRANSITION_TABLE[16]  [static]
```

Initial value:

```
= {
        [(( 0 ) « 2) |  1 )] = +1,
        [(( 1 ) « 2) |  3 )] = +1,
        [(( 3 ) « 2) |  2 )] = +1,
        [(( 2 ) « 2) |  0 )] = +1,
        [(( 1 ) « 2) |  0 )] = -1,
        [(( 3 ) « 2) |  1 )] = -1,
        [(( 2 ) « 2) |  3 )] = -1,
        [(( 0 ) « 2) |  2 )] = -1,
}
```

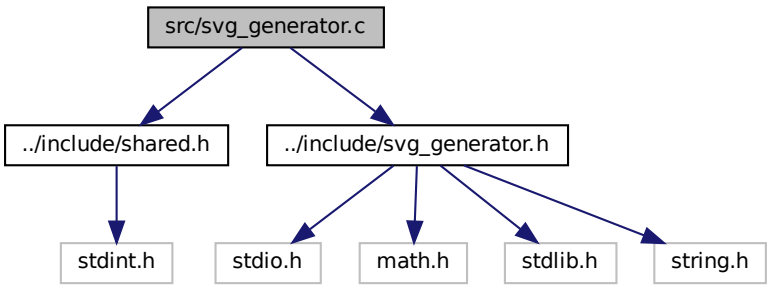
Definition at line 20 of file stepper_motor.c.

Referenced by motor_isr().

5.13 src/svg_generator.c File Reference

```
#include "../include/shared.h"
#include "../include/svg_generator.h"
```

Include dependency graph for svg_generator.c:



Functions

- void generate_html_file (FILE *f)

  Write html + svg plots to the file.
- void svg_tag_start (svg_context *ctx, const char *name, int use_opt)
- void svg_finish (svg_context *ctx)
- void svg_opt_end (svg_context *ctx, int selfclose)
- void svg_optf (svg_context *ctx, const char *key, float v)
- void svg_opti (svg_context *ctx, const char *key, int v)
- void svg_opta (svg_context *ctx, const char *key, const char *v)
- void svg_opt_fill (svg_context *ctx)
- void svg_opt_fill_none (svg_context *ctx)
- void svg_opt_stroke (svg_context *ctx)
- void svg_opt_long_beg (svg_context *ctx, const char *key)
- void svg_opt_long_end (svg_context *ctx)
- void svg_rotate (svg_context *ctx, int angle, int cx, int cy)
- void svg_translate (svg_context *ctx, int dx, int dy)
- void svg_scale (svg_context *ctx, float sx, float sy)
- void svg_tag_end (svg_context *ctx, const char *name)
- void svg_line (svg_context *ctx, float x1, float y1, float x2, float y2)
- void svg_arrow (svg_context *ctx, float x1, float y1, float x2, float y2)
- void svg_path_data (svg_context *ctx, const float *x, const float *y, size_t count)
- void svg_rect (svg_context *ctx, float x, float y, float w, float h)
- void svg_text (svg_context *ctx, const char *txt, float x, float y)
- void svg_set_fill (svg_context *ctx, const char *color)
- void svg_set_dash (svg_context *ctx, const char *dash)
- void svg_set_stroke (svg_context *ctx, const char *color)
- void max_min (const float *data, float *min, float *max, size_t count)
- void svg_draw_to_file_xy (FILE *f, const char *color, float *x, float *y, size_t count, const char *x_label, const char *y_label)

5.13.1 Function Documentation

5.13.1.1 generate_html_file()

```
void generate_html_file (
            FILE * f )
```

Write html + svg plots to the file.

Function for processing data from motors and writing it as html file with svg plots It will display 3 plots: Actual motor position (absolute value) Desire motor position (absolute value) Pwm duty cycle (+- 100%)

The graphs should show at least 2 seconds of history with time resolution 5 ms.

Parameters

desc	: FILE* - file descriptor

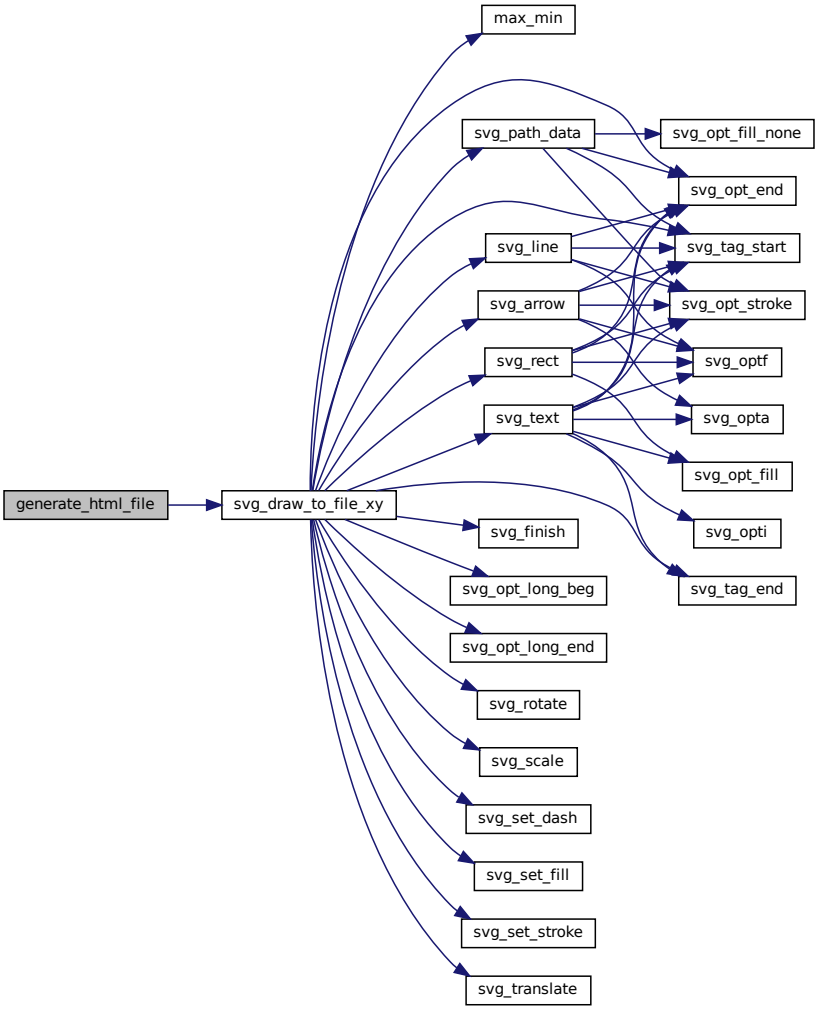
Definition at line 22 of file svg_generator.c.

```
22                                  {
23     float x_actual[HISTORY_SIZE];
24     float y_actual[HISTORY_SIZE];
25     float y_desired[HISTORY_SIZE];
26     float y_pwm[HISTORY_SIZE];
27     int i;
28     uint32_t s = motor_history.array_start_index;
29     int copy_array_idx = 0;
30     for (i = s; i != (s + HISTORY_SIZE) % HISTORY_CYCLIC_ARRAY_SIZE; i = (i + 1) %
31                                                                         HISTORY_CYCLIC_ARRAY_SIZE,
       copy_array_idx++) {
32         y_desired[copy_array_idx] = (float) motor_history.desired_position[i];
33         y_actual[copy_array_idx] = (float) motor_history.position[i];
34         y_pwm[copy_array_idx] = (float) motor_history.pwm_duty[i];
35 
36         x_actual[copy_array_idx] = (float) motor_history.timestamp[i];
37     }
38 
39     // creating svg part
40     svg_draw_to_file_xy(f, "#ff0000", x_actual, y_actual, HISTORY_SIZE, "t", "Actual position (abs
       units)");
41     svg_draw_to_file_xy(f, "#00ff00", x_actual, y_desired, HISTORY_SIZE, "t", "Desired position (abs
       units)");
42     svg_draw_to_file_xy(f, "#0000ff", x_actual, y_pwm, HISTORY_SIZE, "t", "PWM duty cicle, +-100%");
43 }
```

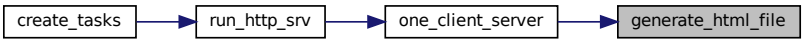
References motor_history_t::array_start_index, motor_history_t::desired_position, HISTORY_CYCLIC_ARRAY_↩ SIZE, HISTORY_SIZE, motor_history, motor_history_t::position, motor_history_t::pwm_duty, svg_draw_to_file_↩ xy(), and motor_history_t::timestamp.

Referenced by one_client_server().

Here is the call graph for this function:



Here is the caller graph for this function:



5.13.1.2 max_min()

```
void max_min (
            const float * data,
```

```
            float * min,
            float * max,
            size_t count )
```

find min and max of an array

Parameters

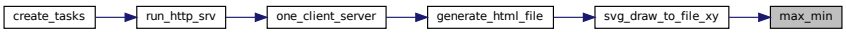
data	float array
min	pointer where to put min
max	pointer where to put max
count	size of an array

Definition at line 307 of file svg_generator.c.

```
307                                                                     {
308     size_t i;
309     for (i = 0; i < count; ++i)
310         if (i == 0) {
311             *min = data[i];
312             *max = data[i];
313         } else if (data[i] < *min)
314             *min = data[i];
315         else if (data[i] > *max)
316             *max = data[i];
317 }
```

Referenced by svg_draw_to_file_xy().

Here is the caller graph for this function:



5.13.1.3 svg_arrow()

```
void svg_arrow (
            svg_context * ctx,
            float x1,
            float y1,
            float x2,
            float y2 )
```

Add an arrow to svg

Parameters

ctx	- pointer to main svg context data structure
x1	- x coor of position where to put object (beginning)
y1	- y coor of position where to put object (beginning)
x2	- x coor of position where to put object (end)
y2	- y coor of position where to put object (end)

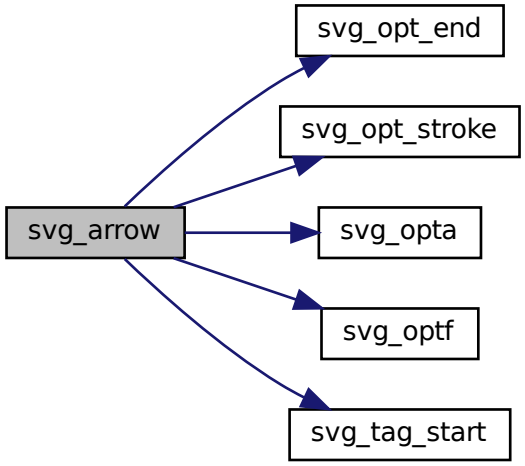
Definition at line 214 of file svg_generator.c.

```
214                                                                                 {
215     svg_tag_start(ctx, "line", 1);
216     {
217         svg_optf(ctx, "x1", cordx(x1));
218         svg_optf(ctx, "y1", cordy(y1));
219         svg_optf(ctx, "x2", cordx(x2));
220         svg_optf(ctx, "y2", cordy(y2));
221         svg_opta(ctx, "marker-end", "url(#markerArrow)");
222         svg_opt_stroke(ctx);
223     }
224     svg_opt_end(ctx, 1);
225 }
```

References cordx, cordy, svg_opt_end(), svg_opt_stroke(), svg_opta(), svg_optf(), and svg_tag_start().

Referenced by svg_draw_to_file_xy().

Here is the call graph for this function:



Here is the caller graph for this function:

5.13.1.4 svg_draw_to_file_xy()

```
void svg_draw_to_file_xy (
            FILE * f,
            const char * color,
            float * x,
```

```
            float * y,
            size_t count,
            const char * x_label,
            const char * y_label )
```

Plot x-y data to SVG file

Parameters

f	opened file for writing
color	line color
y	data values
x	ordered x values from min to max
count	elements count
x_label	label for Ax
y_label	label for Ay

Definition at line 329 of file svg_generator.c.

```
333                                                                     {
334     float min_y, min_x, max_y, max_x;
335     float height, width, t;
336     float mx, my;
337     float tx, ty;
338     char temp[30];
339 //    svg_context *ctx;
340     if (count == 0)return;
341     min_x = x[0];
342     max_x = x[count - 1];
343     /* Find minimum and maximium of Y axis. X mast be already ordered*/
344     max_min(y, &min_y, &max_y, count);
345     width = max_x - min_x;
346     height = max_y - min_y;
347
348     struct svg_context ctx_obj = {
349             .text_size = 14,
350             .stroke_width = 1,
351             .stroke = 1,
352             .fill = 0,
353             .f = f,
354             .width = PLOT_WIDTH,
355             .height = PLOT_HEIGHT,
356             .padding = 50,
357             .flip = 1,
358             .mx = 1,
359             .my = 1,
360             .ofx = 0,
361             .ofy = 0,
362     };
363     svg_context *ctx = &ctx_obj;
364     strcpy(ctx->fill_color, "#ffffff\0");
365     strcpy(ctx->stroke_color, "#000000\0");
366     strcpy(ctx->text_family, "monospace\0");
367     strcpy(ctx->dash_array, "\0");
368
369     /* Begin document */
370
371     fprintf(ctx->f, "<svg version=\"1.2\" width=\"%i\" height=\"%i\"
      xmlns=\"http://www.w3.org/2000/svg\">",
372             ctx->width, ctx->height);
373     mx = (float) (ctx->width - 2 * ctx->padding) / (float) width;
374     my = (float) (ctx->height - 2 * ctx->padding) / (float) height;
375     fprintf(f, "<defs>\n");
376     MARKER_ARROW;
377     fprintf(f, "</defs>\n");
378     svg_set_fill(ctx, "#ffffff");
379     svg_rect(ctx, 0, 0, (float) ctx->width, (float) ctx->height);
380
381     /*Vertical lines*/
382     svg_set_stroke(ctx, "#aaaaaa");
383
384     ctx->stroke_width = 1;
385     for (t = min_x + width / 10; t < max_x; t = t + width / 10) {
386         tx = (t - min_x) * mx + (float) ctx->padding;
387
388         svg_set_dash(ctx, "5,5");
```

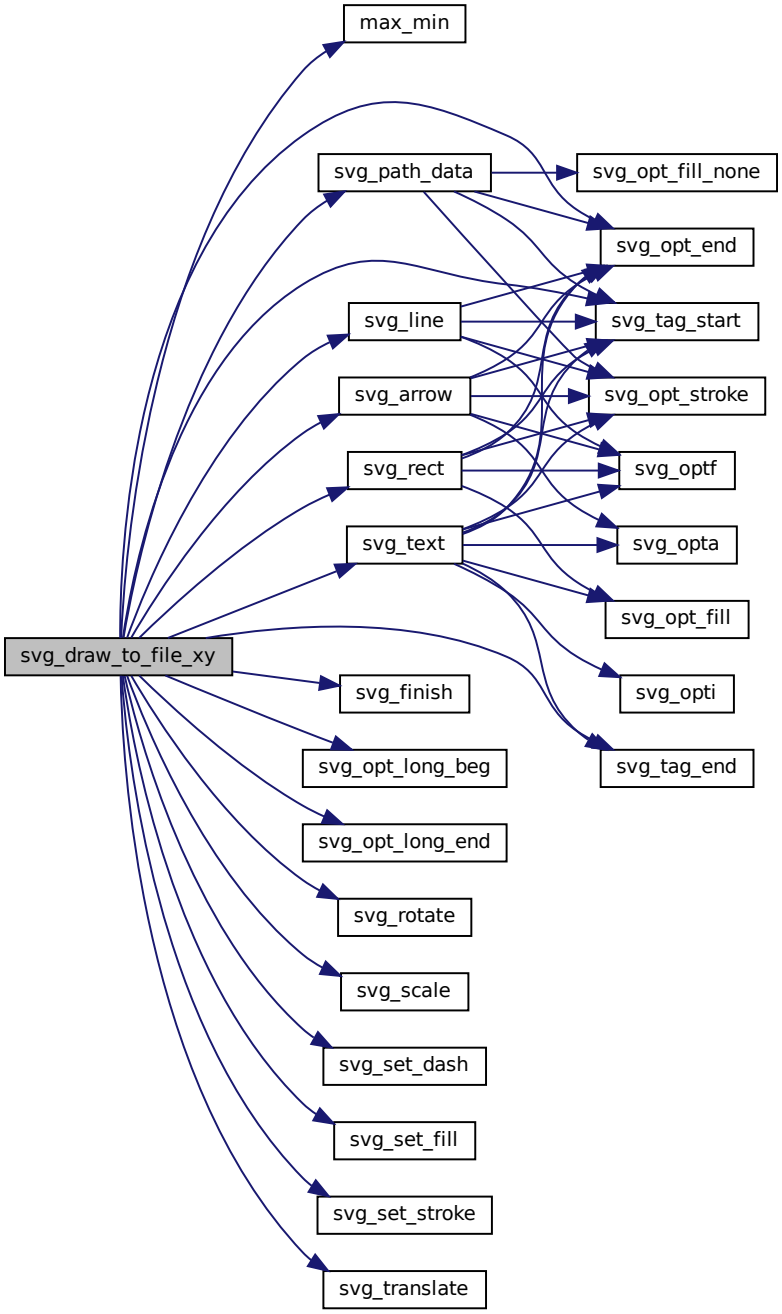
```
389         svg_line(ctx, tx, (float) (ctx->height - ctx->padding), tx, (float) ctx->padding);
390         svg_set_dash(ctx, "");
391         svg_tag_start(ctx, "g", 1);
392         {
393             sprintf(temp, "%i", (int) t);
394             svg_opt_long_beg(ctx, "transform");
395             svg_rotate(ctx, -90, (int) tx - 3, ctx->height - ctx->padding - 10);
396             svg_translate(ctx, (int) tx - 3, ctx->height - ctx->padding - 10);
397             svg_opt_long_end(ctx);
398             svg_opt_end(ctx, 0);
399 
400             svg_text(ctx, temp, 0, 0);
401         }
402         svg_tag_end(ctx, "g");
403     }
404     /*Horizontal lines*/
405     svg_set_stroke(ctx, "#aaaaaa");
406 
407     ctx->stroke_width = 1;
408     for (t = min_y + height / 10; t < max_y; t = t + height / 10.0) {
409         ty = (float) ctx->height - (t - min_y) * my - (float) ctx->padding;
410         svg_set_dash(ctx, "5,5");
411         svg_line(ctx, (float) ctx->padding, ty, (float) (ctx->width - ctx->padding), ty);
412         svg_set_dash(ctx, "");
413         sprintf(temp, "%i", (int) t);
414         svg_text(ctx, temp, (float) ctx->padding + 3, ty - 3);
415     }
416 
417     svg_tag_start(ctx, "g", 1);
418     {
419         ctx->mx = mx;
420         ctx->my = my;
421         ctx->ofx = min_x;
422         ctx->ofy = min_y;
423         svg_opt_long_beg(ctx, "transform");
424         svg_translate(ctx, ctx->padding, ctx->padding);
425         svg_scale(ctx, 1, -1);
426         svg_translate(ctx, 0, -ctx->height + 2 * ctx->padding);
427         svg_opt_long_end(ctx);
428         svg_opt_end(ctx, 0);
429 
430 
431         /* Plot data */
432         svg_set_stroke(ctx, color);
433         svg_set_dash(ctx, "");
434         svg_path_data(ctx, x, y, count);
435         svg_set_stroke(ctx, "#000000");
436 
437     }
438     svg_tag_end(ctx, "g");
439     ctx->mx = 1;
440     ctx->my = 1;
441     ctx->ofx = 0;
442     ctx->ofy = 0;
443     sprintf(temp, "%i", (int) max_y);
444     svg_text(ctx, temp, (float) ctx->padding + 3, (float) ctx->padding);
445     sprintf(temp, "%i", (int) max_x);
446     svg_tag_start(ctx, "g", 1);
447     {
448         svg_opt_long_beg(ctx, "transform");
449         svg_rotate(ctx, -90,
450                    (int) (ctx->width - ctx->padding - ctx->text_size / 2),
451                    (int) (ctx->height - ctx->padding - ctx->text_size));
452         svg_translate(ctx,
453                       (int) (ctx->width - ctx->padding - ctx->text_size),
454                       (int) (ctx->height - ctx->padding - ctx->text_size));
455         svg_opt_long_end(ctx);
456         svg_opt_end(ctx, 0);
457         svg_text(ctx, temp, 0, 0);
458     }
459     svg_tag_end(ctx, "g");
460     /*axis*/
461     svg_set_fill(ctx, "#000000");
462     svg_set_stroke(ctx, "#000000");
463     svg_set_dash(ctx, "");
464     ctx->stroke_width = 2;
465     ctx->text_size = (ctx->padding) / 2;
466 
467     /* Y */
468     svg_set_stroke(ctx, "#000000");
469     svg_set_dash(ctx, "");
470 
471     svg_arrow(ctx,
472               (float) (ctx->padding),
473               (float) (ctx->height - ctx->padding / 2.0),
474               (float) (ctx->padding),
475               (float) (ctx->padding / 2.0));
```

```
476     ctx->stroke_width = 1;
477     svg_text(ctx, y_label,
478              (float) (ctx->padding + ctx->text_size / 4.0),
479              (float) (ctx->padding / 2.0 + ctx->text_size / 4.0));
480
481     sprintf(temp, "%i", (int) x[0]);
482
483     ctx->text_size = 14;
484     svg_tag_start(ctx, "g", 1);
485     {
486         svg_opt_long_beg(ctx, "transform");
487         svg_rotate(ctx, -90, (int) (ctx->padding - ctx->text_size / 2),
488                    (int) (ctx->height - ctx->padding - ctx->text_size));
489         svg_translate(ctx, (int) (ctx->padding - ctx->text_size),
490                       (int) (ctx->height - ctx->padding - ctx->text_size));
491         svg_opt_long_end(ctx);
492         svg_opt_end(ctx, 0);
493         svg_text(ctx, temp, 0, 0);
494     }
495     svg_tag_end(ctx, "g");
496     ctx->text_size = ctx->padding / 2;
497
498     ctx->stroke_width = 2;
499
500     /* X */
501     svg_arrow(ctx,
502               (float) ctx->padding / 2,
503               (float) (ctx->height - ctx->padding),
504               (float) (ctx->width - ctx->padding / 2.0),
505               (float) (ctx->height - ctx->padding));
506     ctx->stroke_width = 1;
507     ctx->text_size = ctx->padding / 2;
508
509     svg_text(ctx, x_label,
510              (float) (ctx->width - strlen(x_label) * ctx->text_size),
511              (float) (ctx->height - ctx->padding - ctx->text_size / 2.0));
512     sprintf(temp, "%i", (int) min_y);
513     ctx->text_size = 14;
514     svg_text(ctx, temp, (float) ctx->padding + 3, (float) (ctx->height - ctx->padding +
    ctx->text_size));
515
516     svg_finish(ctx);
517 }
```

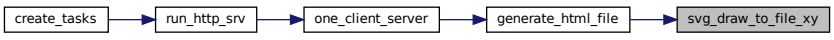
References svg_context::dash_array, svg_context::f, svg_context::fill_color, svg_context::height, MARKER_AR↩ ROW, max_min(), svg_context::mx, svg_context::my, svg_context::ofx, svg_context::ofy, svg_context::padding, PLOT_HEIGHT, PLOT_WIDTH, svg_context::stroke_color, svg_context::stroke_width, svg_arrow(), svg_finish(), svg_line(), svg_opt_end(), svg_opt_long_beg(), svg_opt_long_end(), svg_path_data(), svg_rect(), svg_rotate(), svg_scale(), svg_set_dash(), svg_set_fill(), svg_set_stroke(), svg_tag_end(), svg_tag_start(), svg_text(), svg_↩ translate(), svg_context::text_family, svg_context::text_size, and svg_context::width.

Referenced by generate_html_file().

Here is the call graph for this function:



Here is the caller graph for this function:



5.13.1.5 svg_finish()

```
void svg_finish (
            svg_context * ctx )
```

add svg closing tag

Parameters

ctx	- pointer to main svg context data structure

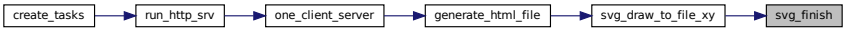
Definition at line 59 of file svg_generator.c.

```
59                                  {
60     fprintf(ctx->f, "</svg>");
61 }
```

References svg_context::f.

Referenced by svg_draw_to_file_xy().

Here is the caller graph for this function:



5.13.1.6 svg_line()

```
void svg_line (
            svg_context * ctx,
            float x1,
            float y1,
            float x2,
            float y2 )
```

Add a line to svg

Parameters

ctx	- pointer to main svg context data structure
x1	- x coor of position where to put object (beginning)
y1	- y coor of position where to put object (beginning)
x2	- x coor of position where to put object (end)
y2	- y coor of position where to put object (end)

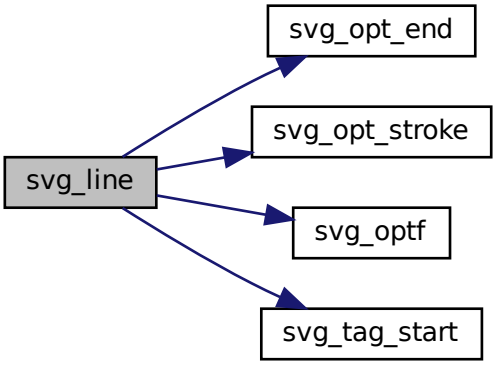
Definition at line 194 of file svg_generator.c.

```
194                                                                     {
195     svg_tag_start(ctx, "line", 1);
196     {
197         svg_optf(ctx, "x1", cordx(x1));
198         svg_optf(ctx, "y1", cordy(y1));
199         svg_optf(ctx, "x2", cordx(x2));
200         svg_optf(ctx, "y2", cordy(y2));
201         svg_opt_stroke(ctx);
202     }
203     svg_opt_end(ctx, 1);
204 }
```

References cordx, cordy, svg_opt_end(), svg_opt_stroke(), svg_optf(), and svg_tag_start().

Referenced by svg_draw_to_file_xy().

Here is the call graph for this function:



Here is the caller graph for this function:

5.13.1.7 svg_opt_end()

```
void svg_opt_end (
            svg_context * ctx,
            int selfclose )
```

add end tag symbol

Parameters

ctx	- pointer to main svg context data structure
selfclose	- 1 if tag is self closing, 0 otherwise

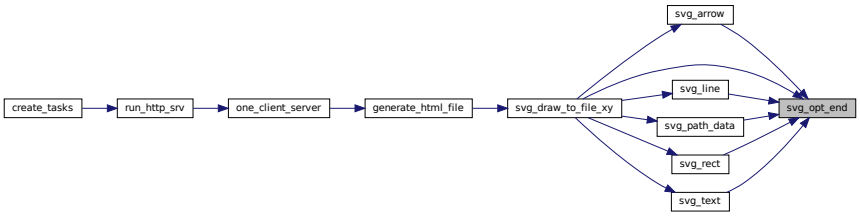
Definition at line 68 of file svg_generator.c.

```
68                                                  {
69     fprintf(ctx->f, " %s>\n", (selfclose ? "/" : ""));
70 }
```

References svg_context::f.

Referenced by svg_arrow(), svg_draw_to_file_xy(), svg_line(), svg_path_data(), svg_rect(), and svg_text().

Here is the caller graph for this function:



5.13.1.8 svg_opt_fill()

```
void svg_opt_fill (
            svg_context * ctx )
```

add fill option to svg tag (using fill_color option from ctx)

Parameters

ctx	- pointer to main svg context data structure

Definition at line 106 of file svg_generator.c.

```
106                                         {
107     fprintf(ctx->f, "fill=\"%s\" ", ctx->fill_color);
108 }
```

References svg_context::f, and svg_context::fill_color.

Referenced by svg_rect(), and svg_text().

Here is the caller graph for this function:

5.13.1.9 svg_opt_fill_none()

```
void svg_opt_fill_none (
            svg_context * ctx )
```

add fill=none option to svg tag

Parameters

ctx	- pointer to main svg context data structure

Definition at line 114 of file svg_generator.c.

```
114                                         {
115     fprintf(ctx->f, "fill=\"none\" ");
116 }
```

References svg_context::f.

Referenced by svg_path_data().

Here is the caller graph for this function:

5.13.1.10 svg_opt_long_beg()

```
void svg_opt_long_beg (
            svg_context * ctx,
            const char * key )
```

open a long option svg tag

Parameters

ctx	- pointer to main svg context data structure
key	- option name

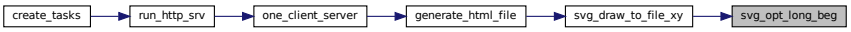
Definition at line 134 of file svg_generator.c.

```
134                                                 {
135     fprintf(ctx->f, "%s=\"", key);
136 }
```

References svg_context::f.

Referenced by svg_draw_to_file_xy().

Here is the caller graph for this function:



5.13.1.11 svg_opt_long_end()

```
void svg_opt_long_end (
            svg_context * ctx )
```

close a long option svg tag

Parameters

ctx	- pointer to main svg context data structure

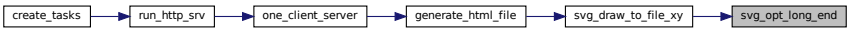
Definition at line 142 of file svg_generator.c.

```
142                                         {
143     fprintf(ctx->f, "\" ");
144 }
```

References svg_context::f.

Referenced by svg_draw_to_file_xy().

Here is the caller graph for this function:



5.13.1.12 svg_opt_stroke()

```
void svg_opt_stroke (
            svg_context * ctx )
```

add svg 'stroke' tag of ctx->stroke_color with ctx->stroke_width and ctx->dash_array

Parameters

ctx	- pointer to main svg context data structure

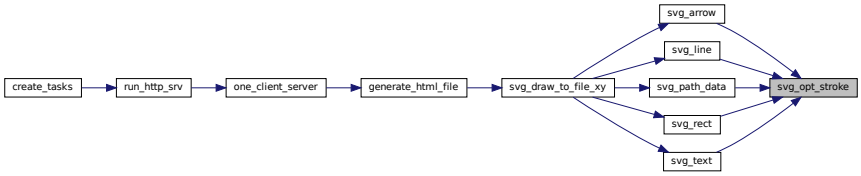
Definition at line 122 of file svg_generator.c.

```
122                                     {
123     fprintf(ctx->f, "stroke=\"%s\" stroke-width=\"%i\" stroke-dasharray=\"%s\" ",
124             ctx->stroke_color,
125             ctx->stroke_width,
126             ctx->dash_array);
127 }
```

References svg_context::dash_array, svg_context::f, svg_context::stroke_color, and svg_context::stroke_width.

Referenced by svg_arrow(), svg_line(), svg_path_data(), svg_rect(), and svg_text().

Here is the caller graph for this function:



5.13.1.13 svg_opta()

```
void svg_opta (
            svg_context * ctx,
            const char * key,
            const char * v )
```

add string option to svg tag

Parameters

ctx	- pointer to main svg context data structure
key	- option name
v	- option value

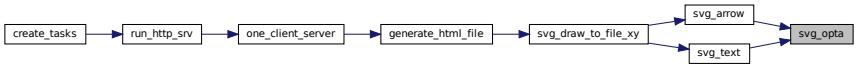
Definition at line 98 of file svg_generator.c.

```
98                                                      {
99     fprintf(ctx->f, "%s=\"%s\" ", key, v);
100 }
```

References svg_context::f.

Referenced by svg_arrow(), and svg_text().

Here is the caller graph for this function:



5.13.1.14 svg_optf()

```
void svg_optf (
            svg_context * ctx,
            const char * key,
            float v )
```

add float option to svg tag

Parameters

ctx	- pointer to main svg context data structure
key	- option name
v	- option value

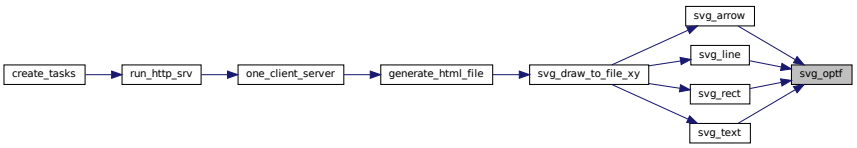
Definition at line 78 of file svg_generator.c.

```
78                                                      {
79     fprintf(ctx->f, "%s=\"%f\" ", key, v);
80 }
```

References svg_context::f.

Referenced by svg_arrow(), svg_line(), svg_rect(), and svg_text().

Here is the caller graph for this function:



5.13.1.15 svg_opti()

```
void svg_opti (
            svg_context * ctx,
            const char * key,
            int v )
```

add int option to svg tag

Parameters

ctx	- pointer to main svg context data structure
key	- option name
v	- option value

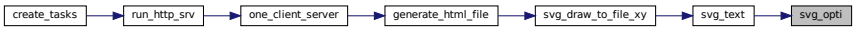
Definition at line 88 of file svg_generator.c.

```
88                                                          {
89     fprintf(ctx->f, "%s=\"%i\" ", key, v);
90 }
```

References svg_context::f.

Referenced by svg_text().

Here is the caller graph for this function:



5.13.1.16 svg_path_data()

```
void svg_path_data (
            svg_context * ctx,
            const float * x,
            const float * y,
            size_t count )
```

Parameters

ctx	- pointer to main svg context data structure
x	- x coor of position where to put object
y	- y coor of position where to put object
count	

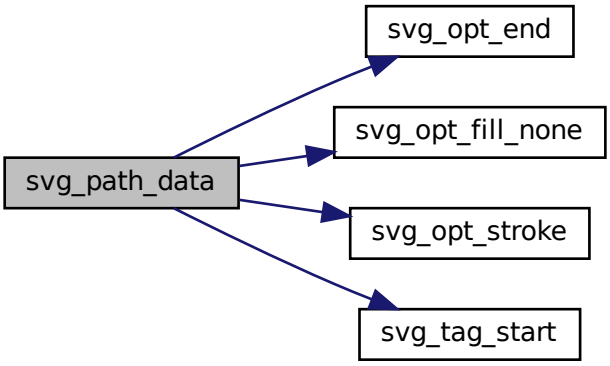
Definition at line 234 of file svg_generator.c.

```
234                                                                                                 {
235     size_t i;
236     svg_tag_start(ctx, "path", 1);
237     {
238         fprintf(ctx->f, " d=\"");
239         for (i = 0; i < count; ++i) {
240             fprintf(ctx->f, "%s%f,%f\n", (i == 0 ? "M" : " L"), cordx(x[i]), cordy(y[i]));
241         }
242         fprintf(ctx->f, "\" ");
243         svg_opt_fill_none(ctx);
244         svg_opt_stroke(ctx);
245     }
246     svg_opt_end(ctx, 1);
247 }
```

References cordx, cordy, svg_context::f, svg_opt_end(), svg_opt_fill_none(), svg_opt_stroke(), and svg_tag_start().

Referenced by svg_draw_to_file_xy().

Here is the call graph for this function:



Here is the caller graph for this function:

5.13.1.17 svg_rect()

```
void svg_rect (
            svg_context * ctx,
            float x,
            float y,
            float w,
            float h )
```

Add rectangle to svg

Parameters

ctx	- pointer to main svg context data structure
x	- x coor of position where to put object
y	- y coor of position where to put object
w	- width of the object, in px
h	- height of the object, in px

Definition at line 257 of file svg_generator.c.

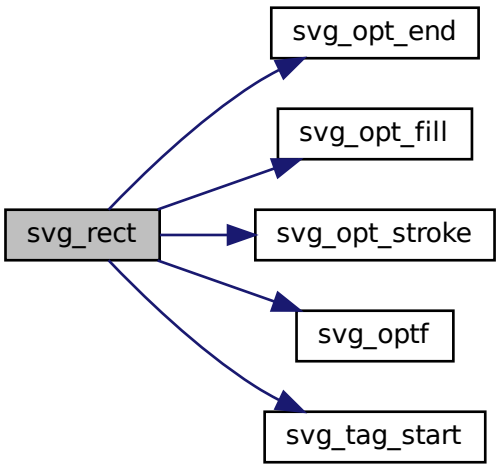
```
257                                                 {
```

```
258     svg_tag_start(ctx, "rect", 1);
259     svg_optf(ctx, "x", cordx(x));
260     svg_optf(ctx, "y", cordy(y));
261     svg_optf(ctx, "width", w);
262     svg_optf(ctx, "height", h);
263     svg_opt_fill(ctx);
264     svg_opt_stroke(ctx);
265     svg_opt_end(ctx, 1);
266 }
```

References cordx, cordy, svg_opt_end(), svg_opt_fill(), svg_opt_stroke(), svg_optf(), and svg_tag_start().

Referenced by svg_draw_to_file_xy().

Here is the call graph for this function:



Here is the caller graph for this function:

5.13.1.18 svg_rotate()

```
void svg_rotate (
            svg_context * ctx,
            int angle,
            int cx,
            int cy )
```

Add x and y rotation to svg tag

Parameters

ctx	- pointer to main svg context data structure
angle	- rotation angle
cx	- rotation in x coor, px
cy	- rotation in y coor, px

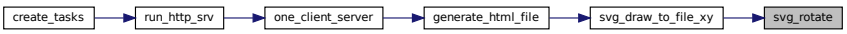
Definition at line 153 of file svg_generator.c.

```
153                                                                 {
154     fprintf(ctx->f, "translate(%i,%i) rotate(%i) translate(%i,%i) ", cx, cy, angle, -cx, -cy);
155 }
```

References svg_context::f.

Referenced by svg_draw_to_file_xy().

Here is the caller graph for this function:



5.13.1.19 svg_scale()

```
void svg_scale (
            svg_context * ctx,
            float sx,
            float sy )
```

Add x and y scaling to svg tag

Parameters

ctx	- pointer to main svg context data structure
sx	- x scaling coefficient
sy	- y scaling coefficient

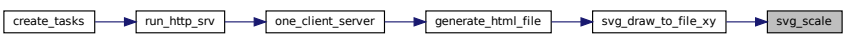
Definition at line 173 of file svg_generator.c.

```
173                                                         {
174     fprintf(ctx->f, "scale(%f,%f) ", sx, sy);
175 }
```

References svg_context::f.

Referenced by svg_draw_to_file_xy().

Here is the caller graph for this function:



5.13.1.20 svg_set_dash()

```
void svg_set_dash (
            svg_context * ctx,
            const char * dash )
```

Definition at line 292 of file svg_generator.c.

```
292                                                     {
293     strcpy(ctx->dash_array, dash);
294 }
```

References svg_context::dash_array.

Referenced by svg_draw_to_file_xy().

Here is the caller graph for this function:

5.13.1.21 svg_set_fill()

```
void svg_set_fill (
            svg_context * ctx,
            const char * color )
```

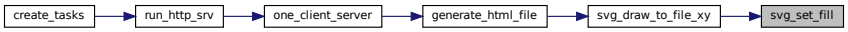
Definition at line 288 of file svg_generator.c.

```
288                                                     {
289     strcpy(ctx->fill_color, color);
290 }
```

References svg_context::fill_color.

Referenced by svg_draw_to_file_xy().

Here is the caller graph for this function:



5.13.1.22 svg_set_stroke()

```
void svg_set_stroke (
            svg_context * ctx,
            const char * color )
```

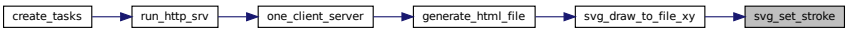
Definition at line 296 of file svg_generator.c.

```
296                                                     {
297     strcpy(ctx->stroke_color, color);
298 }
```

References svg_context::stroke_color.

Referenced by svg_draw_to_file_xy().

Here is the caller graph for this function:



5.13.1.23 svg_tag_end()

```
void svg_tag_end (
            svg_context * ctx,
            const char * name )
```

Close the svg tag

Parameters

ctx	- pointer to main svg context data structure
name	- pointer to the name of the tag to be closed

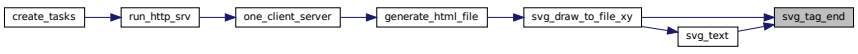
Definition at line 182 of file svg_generator.c.

```
182                                                 {
183     fprintf(ctx->f, "\n</%s>", name);
184 }
```

References svg_context::f.

Referenced by svg_draw_to_file_xy(), and svg_text().

Here is the caller graph for this function:



5.13.1.24 svg_tag_start()

```
void svg_tag_start (
            svg_context * ctx,
            const char * name,
            int use_opt )
```

open svg tag

Parameters

ctx	- pointer to main svg context data structure
name	- opening tag's name
use_opt	- 1 if there are some options in opening tag or not, 0 otherwise

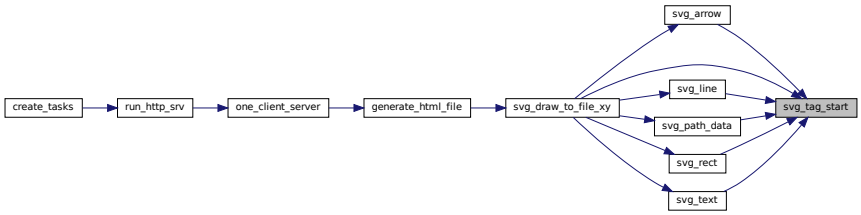
Definition at line 51 of file svg_generator.c.

```
51                                                                  {
52     fprintf(ctx->f, "\n<%s %s", name, (use_opt ? " " : ">"));
53 }
```

References svg_context::f.

Referenced by svg_arrow(), svg_draw_to_file_xy(), svg_line(), svg_path_data(), svg_rect(), and svg_text().

Here is the caller graph for this function:



5.13.1.25 svg_text()

```
void svg_text (
            svg_context * ctx,
            const char * txt,
            float x,
            float y )
```

Add text to svg

Parameters

ctx	- pointer to main svg context data structure
txt	- pointer to text to put it on the svg
x	- x coor of position where to put text
y	- y coor of position where to put text

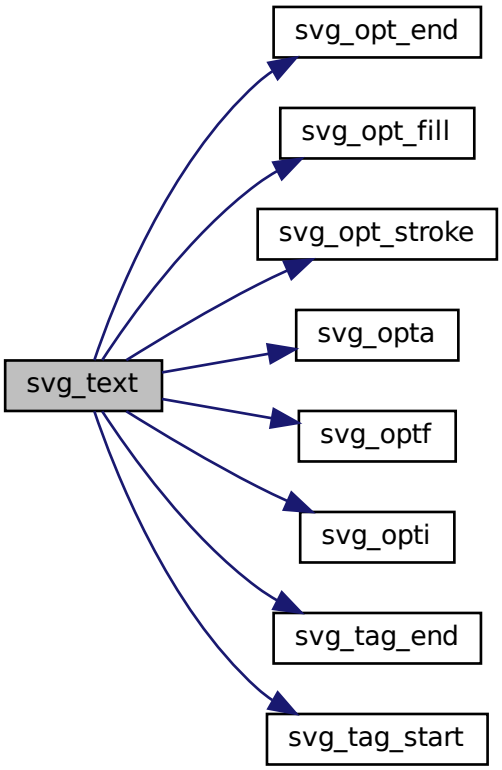
Definition at line 275 of file svg_generator.c.

```
275                                                                    {
276     svg_tag_start(ctx, "text", 1);
277     svg_optf(ctx, "x", cordx(x));
278     svg_optf(ctx, "y", cordy(y));
279     svg_opti(ctx, "font-size", (int) ctx->text_size);
280     svg_opta(ctx, "font-family", ctx->text_family);
281     svg_opt_fill(ctx);
282     svg_opt_stroke(ctx);
283     svg_opt_end(ctx, 0);
284     fprintf(ctx->f, "%s", txt);
285     svg_tag_end(ctx, "text");
286 }
```

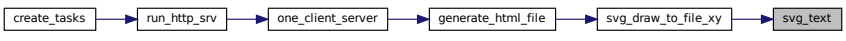
References cordx, cordy, svg_context::f, svg_opt_end(), svg_opt_fill(), svg_opt_stroke(), svg_opta(), svg_optf(), svg_opti(), svg_tag_end(), svg_tag_start(), svg_context::text_family, and svg_context::text_size.

Referenced by svg_draw_to_file_xy().

Here is the call graph for this function:



Here is the caller graph for this function:



5.13.1.26 svg_translate()

```
void svg_translate (
            svg_context * ctx,
            int dx,
            int dy )
```

Add x and y translations to svg tag

Parameters

ctx	- pointer to main svg context data structure
dx	- translation in x coor, px
dy	- translation in y coor, px

Definition at line 163 of file svg_generator.c.

```
163                                                 {
164     fprintf(ctx->f, "translate(%i,%i) ", dx, dy);
165 }
```

References svg_context::f.

Referenced by svg_draw_to_file_xy().

Here is the caller graph for this function: