Watering system with Arudino UNO without Arduino libraries

Generated by Doxygen 1.9.4

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Module Documentation

4.1 GPIO Library <gpio.h>

GPIO library for AVR-GCC.

Functions

- void GPIO_mode_output (volatile uint8_t *reg, uint8_t pin)
  Configure one output pin.
- void GPIO_mode_input_pullup (volatile uint8_t *reg, uint8_t pin)
  Configure one input pin and enable pull-up.
- void GPIO_write_low (volatile uint8_t *reg, uint8_t pin)
  Write one pin to low value.
- void GPIO_write_high (volatile uint8_t *reg, uint8_t pin)
  Write one pin to high value.
- void GPIO_write (volatile uint8_t *reg, uint8_t pin, uint8_t value)
  Write one pin to specific value.
- uint8_t GPIO_read (volatile uint8_t *reg, uint8_t pin)
  Read a value from input pin.

4.1.1 Detailed Description

GPIO library for AVR-GCC.
`#include <gpio.h>`

The library contains functions for controlling AVRs' gpio pin(s).

Note

Based on AVR Libc Reference Manual. Tested on ATmega328P (Arduino Uno), 16 MHz, AVR 8-bit Toolchain 3.6.2.

Author

Tomas Fryza, Dept. of Radio Electronics, Brno University of Technology, Czechia

Copyright

4.1.2 Function Documentation

4.1.2.1 GPIO_mode_input_pullup()

```
void GPIO_mode_input_pullup (
            volatile uint8_t * reg,
            uint8_t pin )
```

Configure one input pin and enable pull-up.

Parameters

reg	Address of Data Direction Register, such as &DDRB
pin	Pin designation in the interval 0 to 7

Returns

none

```
40 {
41     *reg = *reg & ~(1«pin);  // Data Direction Register
42     reg++;                    // Change pointer to Data Register
43     *reg = *reg | (1«pin);   // Data Register
44 }
```

4.1.2.2 GPIO_mode_output()

```
void GPIO_mode_output (
            volatile uint8_t * reg,
            uint8_t pin )
```

Configure one output pin.

Parameters

reg	Address of Data Direction Register, such as &DDRB
pin	Pin designation in the interval 0 to 7

Returns

none

```
27 {
28     *reg = *reg | (1«pin);
29 }
```

Referenced by main().

Here is the caller graph for this function:

4.1.2.3 GPIO_read()

```
uint8_t GPIO_read (
              volatile uint8_t * reg,
              uint8_t pin )
```

Read a value from input pin.

Parameters

reg	Address of Pin Register, such as &PINB
pin	Pin designation in the interval 0 to 7

Returns

Pin value

```
95 {
96     uint8_t temp;
97
98     temp = *reg & (1<<pin);
99
100     if (temp != 0) {
101         return 1;
102     }
103     else {
104         return 0;
105     }
106 }
```

4.1.2.4 GPIO_write()

```
void GPIO_write (
              volatile uint8_t * reg,
              uint8_t pin,
              uint8_t value )
```

Write one pin to specific value.

Parameters

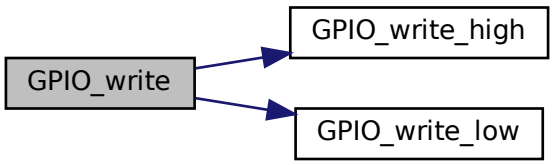
reg	Address of Port Register, such as &PORTB
pin	Pin designation in the interval 0 to 7
value	0=LOW, 1=HIGH

Returns

none

```
81 {
82     if(value) GPIO_write_high(reg, pin);
83     else GPIO_write_low(reg, pin);
84 }
```

Here is the call graph for this function:



4.1.2.5 GPIO_write_high()

```
void GPIO_write_high (
            volatile uint8_t * reg,
            uint8_t pin )
```

Write one pin to high value.

Parameters

reg	Address of Port Register, such as &PORTB
pin	Pin designation in the interval 0 to 7

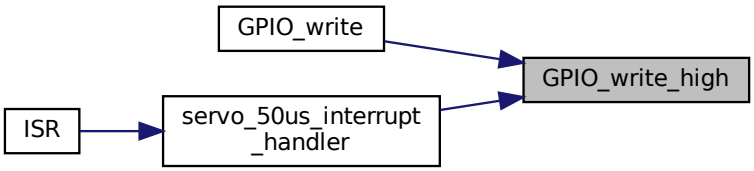
Returns

none

```
68 {
69     *reg = *reg | (1«pin);
70 }
```

Referenced by GPIO_write(), and servo_50us_interrupt_handler().

Here is the caller graph for this function:



4.1.2.6 GPIO_write_low()

```
void GPIO_write_low (
            volatile uint8_t * reg,
            uint8_t pin )
```

Write one pin to low value.

Parameters

reg	Address of Port Register, such as &PORTB
pin	Pin designation in the interval 0 to 7

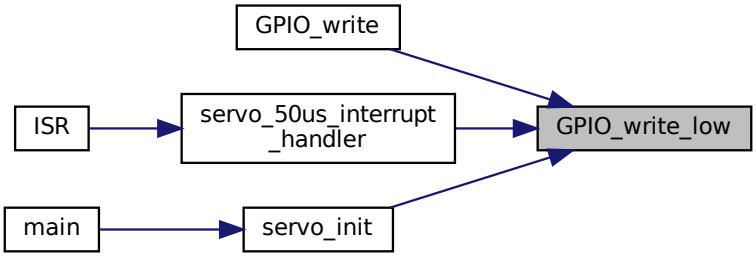
Returns

none

```
55 {
56     *reg = *reg & ~(1<<pin);
57 }
```

Referenced by GPIO_write(), servo_50us_interrupt_handler(), and servo_init().

Here is the caller graph for this function:



4.2 Timer Library <timer.h>

Timer library for AVR-GCC.

Macros

- #define TCCRXB_MODIFY_CS(reg, val) reg = (reg & 0b111) | val

Definitions for 16-bit Timer/Counter1

Note

t_OVF = 1/F_CPU * prescaler * 2^n where n = 16, F_CPU = 16 MHz

- #define TIM0_STOP() TCCRXB_MODIFY_CS(TCCR0B, 0)

  Stop timer, prescaler 000 --> STOP.
- #define TIM0_OVF_16US() TCCRXB_MODIFY_CS(TCCR0B, 1)

  Set overflow 16us, prescaler 001 --> 1.
- #define TIM0_OVF_128US() TCCRXB_MODIFY_CS(TCCR0B, 2)

  Set overflow ms, prescaler 010 --> 8.
- #define TIM0_OVF_1MS() TCCRXB_MODIFY_CS(TCCR0B, 3)

  Set overflow 1ms, prescaler 011 --> 64.
- #define TIM0_OVF_4MS() TCCRXB_MODIFY_CS(TCCR0B, 3)

  Set overflow 4ms, 1024 prescaler 100 --> 256.
- #define TIM0_OVF_16MS() TCCRXB_MODIFY_CS(TCCR0B, 5)

  Set overflow 16ms, prescaler // 101 --> 1024.
- #define TIM1_STOP() TCCRXB_MODIFY_CS(TCCR1B, 0)

  Stop timer, prescaler 000 --> STOP.
- #define TIM1_OVF_4MS() TCCRXB_MODIFY_CS(TCCR1B, 1)

  Set overflow 4ms, prescaler 001 --> 1.
- #define TIM1_OVF_33MS() TCCRXB_MODIFY_CS(TCCR1B, 2)

  Set overflow 33ms, prescaler 010 --> 8.
- #define TIM1_OVF_262MS() TCCRXB_MODIFY_CS(TCCR1B, 3)

  Set overflow 262ms, prescaler 011 --> 64.
- #define TIM1_OVF_1SEC() TCCRXB_MODIFY_CS(TCCR1B, 4)

  Set overflow 1s, prescaler 100 --> 256.
- #define TIM1_OVF_4SEC() TCCRXB_MODIFY_CS(TCCR1B, 5)

  Set overflow 4s, prescaler // 101 --> 1024.
- #define TIM2_STOP() TCCRXB_MODIFY_CS(TCCR2B, 0)

  Stop timer, prescaler 000 --> STOP.
- #define TIM2_OVF_16US() TCCRXB_MODIFY_CS(TCCR2B, 1)

  Set overflow 16us, prescaler 001 --> 1.
- #define TIM2_OVF_128US() TCCRXB_MODIFY_CS(TCCR2B, 2)

  Set overflow 128us, prescaler 010 --> 8.
- #define TIM2_OVF_512US() TCCRXB_MODIFY_CS(TCCR2B, 3)

  Set overflow 512, prescaler 011 --> 32.
- #define TIM2_OVF_1MS() TCCRXB_MODIFY_CS(TCCR2B, 4)

  Set overflow 1ms, 1024 prescaler 100 --> 64.
- #define TIM2_OVF_2MS() TCCRXB_MODIFY_CS(TCCR2B, 5)

Set overflow 2ms, prescaler // 101 --> 128.

- #define TIM2_OVF_4MS() TCCRXB_MODIFY_CS(TCCR2B, 6)

  Set overflow 4ms, prescaler // 110 --> 256.
- #define TIM2_OVF_16MS() TCCRXB_MODIFY_CS(TCCR2B, 7)

  Set overflow 16ms, prescaler // 111 --> 1024.
- #define TIM0_OVF_ENABLE() TIMSK0 |= (1<<TOIE0)

  Enable overflow interrupt, 1 --> enable.
- #define TIM0_OVF_DISABLE() TIMSK0 &= ~(1<<TOIE0)

  Disable overflow interrupt, 0 --> disable.
- #define TIM1_OVF_ENABLE() TIMSK1 |= (1<<TOIE1)

  Enable overflow interrupt, 1 --> enable.
- #define TIM1_OVF_DISABLE() TIMSK1 &= ~(1<<TOIE1)

  Disable overflow interrupt, 0 --> disable.
- #define TIM2_OVF_ENABLE() TIMSK2 |= (1<<TOIE2)

  Enable overflow interrupt, 1 --> enable.
- #define TIM2_OVF_DISABLE() TIMSK2 &= ~(1<<TOIE2)

  Disable overflow interrupt, 0 --> disable.

4.2.1 Detailed Description

Timer library for AVR-GCC.
`#include <timer.h>`

The library contains macros for controlling the timer modules.

Note

Based on Microchip Atmel ATmega328P manual and no source file is needed for the library.

Author

Tomas Fryza, Dept. of Radio Electronics, Brno University of Technology, Czechia

Copyright

(c) 2019 Tomas Fryza, This work is licensed under the terms of the MIT license

4.2.2 Macro Definition Documentation

4.2.2.1 TCCRXB_MODIFY_CS

```
#define TCCRXB_MODIFY_CS(
              reg,
              val ) reg = (reg & 0b111) | val
```

4.2.2.2 TIM0_OVF_128US

```
#define TIM0_OVF_128US( ) TCCRXB_MODIFY_CS(TCCR0B, 2)
```

Set overflow ms, prescaler 010 --> 8.

4.2.2.3 TIM0_OVF_16MS

```
#define TIM0_OVF_16MS( ) TCCRXB_MODIFY_CS(TCCR0B, 5)
```

Set overflow 16ms, prescaler // 101 --> 1024.

4.2.2.4 TIM0_OVF_16US

```
#define TIM0_OVF_16US( ) TCCRXB_MODIFY_CS(TCCR0B, 1)
```

Set overflow 16us, prescaler 001 --> 1.

4.2.2.5 TIM0_OVF_1MS

```
#define TIM0_OVF_1MS( ) TCCRXB_MODIFY_CS(TCCR0B, 3)
```

Set overflow 1ms, prescaler 011 --> 64.

4.2.2.6 TIM0_OVF_4MS

```
#define TIM0_OVF_4MS( ) TCCRXB_MODIFY_CS(TCCR0B, 3)
```

Set overflow 4ms, 1024 prescaler 100 --> 256.

4.2.2.7 TIM0_OVF_DISABLE

```
#define TIM0_OVF_DISABLE( ) TIMSK0 &= ~(1<<TOIE0)
```

Disable overflow interrupt, 0 --> disable.

4.2.2.8 TIM0_OVF_ENABLE

```
#define TIM0_OVF_ENABLE( ) TIMSK0 |= (1<<TOIE0)
```

Enable overflow interrupt, 1 --> enable.

4.2.2.9 TIM0_STOP

```
#define TIM0_STOP( ) TCCRXB_MODIFY_CS(TCCR0B, 0)
```

Stop timer, prescaler 000 --> STOP.

4.2.2.10 TIM1_OVF_1SEC

```
#define TIM1_OVF_1SEC( ) TCCRXB_MODIFY_CS(TCCR1B, 4)
```

Set overflow 1s, prescaler 100 --> 256.

4.2.2.11 TIM1_OVF_262MS

```
#define TIM1_OVF_262MS( ) TCCRXB_MODIFY_CS(TCCR1B, 3)
```

Set overflow 262ms, prescaler 011 --> 64.

4.2.2.12 TIM1_OVF_33MS

```
#define TIM1_OVF_33MS( ) TCCRXB_MODIFY_CS(TCCR1B, 2)
```

Set overflow 33ms, prescaler 010 --> 8.

4.2.2.13 TIM1_OVF_4MS

```
#define TIM1_OVF_4MS( ) TCCRXB_MODIFY_CS(TCCR1B, 1)
```

Set overflow 4ms, prescaler 001 --> 1.

4.2.2.14 TIM1_OVF_4SEC

```
#define TIM1_OVF_4SEC( ) TCCRXB_MODIFY_CS(TCCR1B, 5)
```

Set overflow 4s, prescaler // 101 --> 1024.

4.2.2.15 TIM1_OVF_DISABLE

```
#define TIM1_OVF_DISABLE( ) TIMSK1 &= ~(1<<TOIE1)
```

Disable overflow interrupt, 0 --> disable.

4.2.2.16 TIM1_OVF_ENABLE

```
#define TIM1_OVF_ENABLE( ) TIMSK1 |= (1<<TOIE1)
```

Enable overflow interrupt, 1 --> enable.

4.2.2.17 TIM1_STOP

```
#define TIM1_STOP( ) TCCRXB_MODIFY_CS(TCCR1B, 0)
```

Stop timer, prescaler 000 --> STOP.

4.2.2.18 TIM2_OVF_128US

```
#define TIM2_OVF_128US( ) TCCRXB_MODIFY_CS(TCCR2B, 2)
```

Set overflow 128us, prescaler 010 --> 8.

4.2.2.19 TIM2_OVF_16MS

```
#define TIM2_OVF_16MS( ) TCCRXB_MODIFY_CS(TCCR2B, 7)
```

Set overflow 16ms, prescaler // 111 --> 1024.

4.2.2.20 TIM2_OVF_16US

```
#define TIM2_OVF_16US( ) TCCRXB_MODIFY_CS(TCCR2B, 1)
```

Set overflow 16us, prescaler 001 --> 1.

4.2.2.21 TIM2_OVF_1MS

```
#define TIM2_OVF_1MS( ) TCCRXB_MODIFY_CS(TCCR2B, 4)
```

Set overflow 1ms, 1024 prescaler 100 --> 64.

4.2.2.22 TIM2_OVF_2MS

```
#define TIM2_OVF_2MS( ) TCCRXB_MODIFY_CS(TCCR2B, 5)
```

Set overflow 2ms, prescaler // 101 --> 128.

4.2.2.23 TIM2_OVF_4MS

```
#define TIM2_OVF_4MS( ) TCCRXB_MODIFY_CS(TCCR2B, 6)
```

Set overflow 4ms, prescaler // 110 --> 256.

4.2.2.24 TIM2_OVF_512US

```
#define TIM2_OVF_512US( ) TCCRXB_MODIFY_CS(TCCR2B, 3)
```

Set overflow 512, prescaler 011 --> 32.

4.2.2.25 TIM2_OVF_DISABLE

```
#define TIM2_OVF_DISABLE( ) TIMSK2 &= ~(1<<TOIE2)
```

Disable overflow interrupt, 0 --> disable.

4.2.2.26 TIM2_OVF_ENABLE

```
#define TIM2_OVF_ENABLE( ) TIMSK2 |= (1<<TOIE2)
```

Enable overflow interrupt, 1 --> enable.

4.2.2.27 TIM2_STOP

```
#define TIM2_STOP( ) TCCRXB_MODIFY_CS(TCCR2B, 0)
```

Stop timer, prescaler 000 --> STOP.

4.3 TWI Library <twi.h>

I2C/TWI library for AVR-GCC.

Other definitions

- void twi_init (void)

  Initialize TWI unit, enable internal pull-ups, and set SCL frequency.
- void twi_start (void)

  Start communication on I2C/TWI bus.
- uint8_t twi_write (uint8_t data)

  Send one byte to I2C/TWI Slave device.
- uint8_t twi_read (uint8_t ack)

  Read one byte from I2C/TWI Slave device and acknowledge it by ACK or NACK.
- void twi_stop (void)

  Generates Stop condition on I2C/TWI bus.
- uint8_t twi_test_address (uint8_t adr)

  Test presence of one I2C device on the bus.
- #define TWI_WRITE 0

  Mode for writing to I2C/TWI device.
- #define TWI_READ 1

  Mode for reading from I2C/TWI device.
- #define TWI_ACK 0

  ACK value for writing to I2C/TWI bus.
- #define TWI_NACK 1

  NACK value for writing to I2C/TWI bus.
- #define DDR(_x) (*(&_x - 1))

  Address of Data Direction Register of port _x.
- #define PIN(_x) (*(&_x - 2))

  Address of input register of port _x.

Definition of frequencies

- #define F_CPU 16000000

  CPU frequency in Hz required TWI_BIT_RATE_REG.
- #define F_SCL 100000

  I2C/TWI bit rate. Must be greater than 31000.
- #define TWI_BIT_RATE_REG ((F_CPU/F_SCL - 16) / 2)

  TWI bit rate register value.

Definition of ports and pins

- #define TWI_PORT PORTC

  Port of TWI unit.
- #define TWI_SDA_PIN 4

  SDA pin of TWI unit.
- #define TWI_SCL_PIN 5

  SCL pin of TWI unit.

4.3.1 Detailed Description

I2C/TWI library for AVR-GCC.

```
#include <twi.h>
```

This library defines functions for the TWI (I2C) communication between AVR and Slave device(s). Functions use internal TWI module of AVR.

Note

Only Master transmitting and Master receiving modes are implemented. Based on Microchip Atmel ATmega16 and ATmega328P manuals.

Author

Tomas Fryza, Dept. of Radio Electronics, Brno University of Technology, Czechia

Copyright

4.3.2 Macro Definition Documentation

4.3.2.1 DDR

```
#define DDR(
              _x ) (*(&_x - 1))
```

Address of Data Direction Register of port _x.

4.3.2.2 F_CPU

```
#define F_CPU 16000000
```

CPU frequency in Hz required TWI_BIT_RATE_REG.

4.3.2.3 F_SCL

```
#define F_SCL 100000
```

I2C/TWI bit rate. Must be greater than 31000.

4.3.2.4 PIN

```
#define PIN(
             _x ) (*(&_x - 2))
```

Address of input register of port _x.

4.3.2.5 TWI_ACK

```
#define TWI_ACK 0
```

ACK value for writing to I2C/TWI bus.

4.3.2.6 TWI_BIT_RATE_REG

```
#define TWI_BIT_RATE_REG ((F_CPU/F_SCL - 16) / 2)
```

TWI bit rate register value.

4.3.2.7 TWI_NACK

```
#define TWI_NACK 1
```

NACK value for writing to I2C/TWI bus.

4.3.2.8 TWI_PORT

```
#define TWI_PORT PORTC
```

Port of TWI unit.

4.3.2.9 TWI_READ

```
#define TWI_READ 1
```

Mode for reading from I2C/TWI device.

4.3.2.10 TWI_SCL_PIN

```
#define TWI_SCL_PIN 5
```

SCL pin of TWI unit.

4.3.2.11 TWI_SDA_PIN

```
#define TWI_SDA_PIN 4
```

SDA pin of TWI unit.

4.3.2.12 TWI_WRITE

```
#define TWI_WRITE 0
```

Mode for writing to I2C/TWI device.

4.3.3 Function Documentation

4.3.3.1 twi_init()

```
void twi_init (
              void  )
```

Initialize TWI unit, enable internal pull-ups, and set SCL frequency.

Implementation notes:

- AVR internal pull-up resistors at pins TWI_SDA_PIN and TWI_SCL_PIN are enabled
- TWI bit rate register value is calculated as follows fscl = fcpu/(16 + 2∗TWBR)

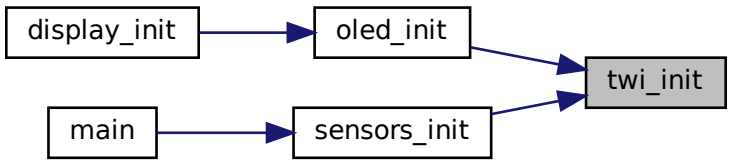
Returns

none

```
29 {
30     /* Enable internal pull-up resistors */
31     DDR(TWI_PORT) &= ~((1«TWI_SDA_PIN) | (1«TWI_SCL_PIN));
32     TWI_PORT |= (1«TWI_SDA_PIN) | (1«TWI_SCL_PIN);
33
34     /* Set SCL frequency */
35     TWSR &= ~((1«TWPS1) | (1«TWPS0));
36     TWBR = TWI_BIT_RATE_REG;
37 }
```

Referenced by oled_init(), and sensors_init().

Here is the caller graph for this function:



4.3.3.2 twi_read()

```
uint8_t twi_read (
              uint8_t ack )
```

Read one byte from I2C/TWI Slave device and acknowledge it by ACK or NACK.

Parameters

ack	- ACK/NACK value to be transmitted

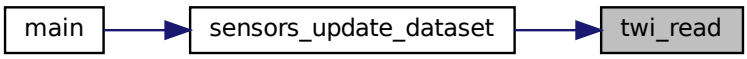
Returns

Received data byte

```
91 {
92     if (ack == TWI_ACK)
93         TWCR = (1«TWINT) | (1«TWEN) | (1«TWEA);
94     else
95         TWCR = (1«TWINT) | (1«TWEN);
96     for(uint32_t i = 0; i < TWI_TIMEOUT && ((TWCR & (1«TWINT)) == 0); i++) asm("NOP");
97
98     return (TWDR);
99 }
```

Referenced by sensors_update_dataset().

Here is the caller graph for this function:



4.3.3.3 twi_start()

```
void twi_start (
             void  )
```

Start communication on I2C/TWI bus.

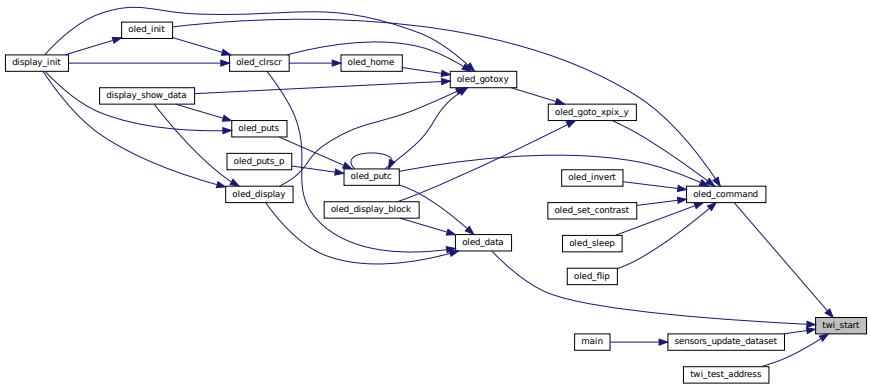
Returns

none

```
46 {
47     /* Send Start condition */
48     TWCR = (1«TWINT) | (1«TWSTA) | (1«TWEN);
49     for(uint32_t i = 0; i < TWI_TIMEOUT && ((TWCR & (1«TWINT)) == 0); i++) asm("NOP");
50 }
```

Referenced by oled_command(), oled_data(), sensors_update_dataset(), and twi_test_address().

Here is the caller graph for this function:



4.3.3.4 twi_stop()

```
void twi_stop (
            void  )
```

Generates Stop condition on I2C/TWI bus.

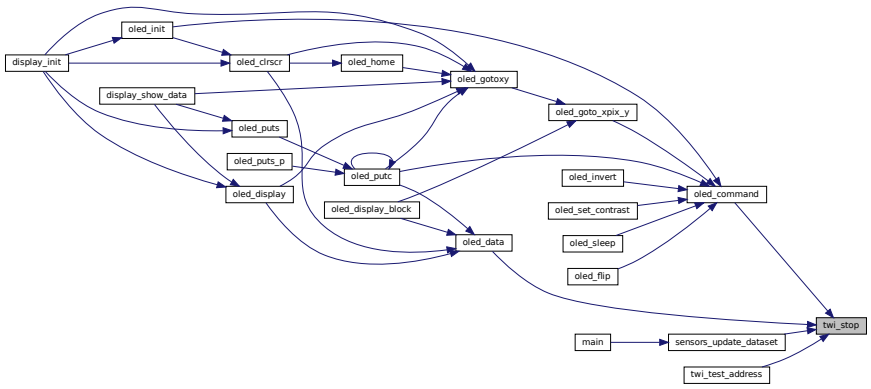
Returns

none

```
108 {
109     /* Generate Stop condition on I2C/TWI bus */
110     TWCR = (1<<TWINT) | (1<<TWSTO) | (1<<TWEN);
111 }
```

Referenced by oled_command(), oled_data(), sensors_update_dataset(), and twi_test_address().

Here is the caller graph for this function:



4.3.3.5 twi_test_address()

```
uint8_t twi_test_address (
            uint8_t adr )
```

Test presence of one I2C device on the bus.

Parameters

adr	Slave address

Returns

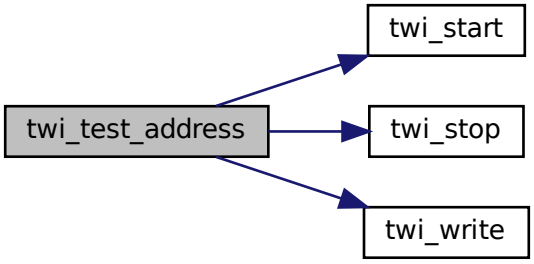
ACK/NACK received value

Return values

0	- ACK has been received
1	- NACK has been received

```
121 {
122     uint8_t ack;  // ACK response from Slave
123
124     twi_start();
125     ack = twi_write((adr<<1) | TWI_WRITE);
126     twi_stop();
127
128     return ack;
129 }
```

Here is the call graph for this function:



4.3.3.6 twi_write()

```
uint8_t twi_write (
              uint8_t data )
```

Send one byte to I2C/TWI Slave device.

Parameters

data	Byte to be transmitted

Returns

ACK/NACK received value

Return values

0	- ACK has been received
1	- NACK has been received

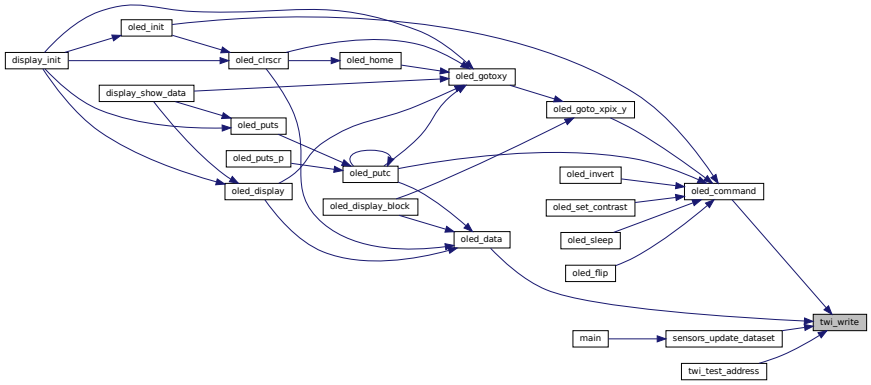
Note

Function returns 0 if 0x18, 0x28, or 0x40 status code is detected
0x18: SLA+W has been transmitted and ACK has been received
0x28: Data byte has been transmitted and ACK has been received
0x40: SLA+R has been transmitted and ACK has been received

```
60 {
61     uint8_t twi_status;
62 
63     /* Send SLA+R, SLA+W, or data byte on I2C/TWI bus */
64     TWDR = data;
65     TWCR = (1«TWINT) | (1«TWEN);
66     for(uint32_t i = 0; i < TWI_TIMEOUT && ((TWCR & (1«TWINT)) == 0); i++) asm("NOP");
67 
68     /* Check value of TWI status register */
69     twi_status = TWSR & 0xf8;
70 
71     /* Status Code:
72  * 0x18:  SLA+W has been transmitted and ACK received
73  * 0x28:  Data byte has been transmitted and ACK has been received
74  * 0x40:  SLA+R has been transmitted and ACK received
75  */
76     if (twi_status == 0x18 || twi_status == 0x28 || twi_status == 0x40)
77         return 0;    /* ACK received */
78     else
79         return 1;    /* NACK received */
80 }
```

Referenced by oled_command(), oled_data(), sensors_update_dataset(), and twi_test_address().

Here is the caller graph for this function:



4.4 UART Library <uart.h>

Interrupt UART library using the built-in UART with transmit and receive circular buffers.

Macros

- #define UART_BAUD_SELECT(baudRate, xtalCpu) (((xtalCpu) + 8UL * (baudRate)) / (16UL * (baudRate)) -1UL)

  UART Baudrate Expression.
- #define UART_BAUD_SELECT_DOUBLE_SPEED(baudRate, xtalCpu) (((((xtalCpu) + 4UL * (baudRate)) / (8UL * (baudRate)) -1UL)) | 0x8000)

  UART Baudrate Expression for ATmega double speed mode.

- #define UART_RX_BUFFER_SIZE 32

  Size of the circular receive buffer, must be power of 2.
- #define UART_TX_BUFFER_SIZE 256

  Size of the circular transmit buffer, must be power of 2.
- #define UART_FRAME_ERROR 0x1000

  Framing Error by UART
- #define UART_OVERRUN_ERROR 0x0800

  Overrun condition by UART
- #define UART_PARITY_ERROR 0x0400

  Parity Error by UART
- #define UART_BUFFER_OVERFLOW 0x0200

  receive ringbuffer overflow
- #define UART_NO_DATA 0x0100

  no receive data available
- #define uart_puts_P(__s) uart_puts_p(PSTR(__s))

  Macro to automatically put a string constant into program memory.
- #define uart1_puts_P(__s) uart1_puts_p(PSTR(__s))

  Macro to automatically put a string constant into program memory.

Functions

- void uart_init (unsigned int baudrate)

  Initialize UART and set baudrate.
- unsigned int uart_getc (void)

  Get received byte from ringbuffer.
- void uart_putc (unsigned char data)

  Put byte to ringbuffer for transmitting via UART.
- void uart_puts (const char *s)

  Put string to ringbuffer for transmitting via UART.
- void uart_puts_p (const char *s)

  Put string from program memory to ringbuffer for transmitting via UART.
- void uart1_init (unsigned int baudrate)

  Initialize USART1 (only available on selected ATmegas)
- unsigned int uart1_getc (void)

  Get received byte of USART1 from ringbuffer. (only available on selected ATmega)
- void uart1_putc (unsigned char data)

  Put byte to ringbuffer for transmitting via USART1 (only available on selected ATmega)
- void uart1_puts (const char *s)

  Put string to ringbuffer for transmitting via USART1 (only available on selected ATmega)
- void uart1_puts_p (const char *s)

  Put string from program memory to ringbuffer for transmitting via USART1 (only available on selected ATmega)

4.4.1 Detailed Description

Interrupt UART library using the built-in UART with transmit and receive circular buffers.
`#include <uart.h>`

This library can be used to transmit and receive data through the built in UART.

An interrupt is generated when the UART has finished transmitting or receiving a byte. The interrupt handling routines use circular buffers for buffering received and transmitted data.

The UART_RX_BUFFER_SIZE and UART_TX_BUFFER_SIZE constants define the size of the circular buffers in bytes. Note that these constants must be a power of 2. You may need to adapt these constants to your target and your application by adding CDEFS += -DUART_RX_BUFFER_SIZE=nn -DUART_TX_BUFFER_SIZE=nn to your Makefile.

Note

Based on Atmel Application Note AVR306

Author

Peter Fleury pfleury@gmx.ch http://tinyurl.com/peterfleury

Copyright

4.4.2 Macro Definition Documentation

4.4.2.1 uart1_puts_P

```
#define uart1_puts_P(
             __s ) uart1_puts_p(PSTR(__s))
```

Macro to automatically put a string constant into program memory.

4.4.2.2 UART_BAUD_SELECT

```
#define UART_BAUD_SELECT(
             baudRate,
             xtalCpu ) (((xtalCpu) + 8UL * (baudRate)) / (16UL * (baudRate)) -1UL)
```

UART Baudrate Expression.

Parameters

xtalCpu	system clock in Mhz, e.g. 4000000UL for 4Mhz
baudRate	baudrate in bps, e.g. 1200, 2400, 9600

4.4.2.3 UART_BAUD_SELECT_DOUBLE_SPEED

```
#define UART_BAUD_SELECT_DOUBLE_SPEED(
            baudRate,
            xtalCpu ) ( ((((xtalCpu) + 4UL * (baudRate)) / (8UL * (baudRate)) -1UL)) | 0x8000)
```

UART Baudrate Expression for ATmega double speed mode.

Parameters

xtalCpu	system clock in Mhz, e.g. 4000000UL for 4Mhz
baudRate	baudrate in bps, e.g. 1200, 2400, 9600

4.4.2.4 UART_BUFFER_OVERFLOW

```
#define UART_BUFFER_OVERFLOW 0x0200
```

receive ringbuffer overflow

4.4.2.5 UART_FRAME_ERROR

```
#define UART_FRAME_ERROR 0x1000
```

Framing Error by UART

4.4.2.6 UART_NO_DATA

```
#define UART_NO_DATA 0x0100
```

no receive data available

4.4.2.7 UART_OVERRUN_ERROR

```
#define UART_OVERRUN_ERROR 0x0800
```

Overrun condition by UART

4.4.2.8 UART_PARITY_ERROR

```
#define UART_PARITY_ERROR 0x0400
```

Parity Error by UART

4.4.2.9 uart_puts_P

```
#define uart_puts_P(
             __s ) uart_puts_p(PSTR(__s))
```

Macro to automatically put a string constant into program memory.

4.4.2.10 UART_RX_BUFFER_SIZE

```
#define UART_RX_BUFFER_SIZE 32
```

Size of the circular receive buffer, must be power of 2.

You may need to adapt this constant to your target and your application by adding CDEFS += -DUART_RX_↩ BUFFER_SIZE=nn to your Makefile.

4.4.2.11 UART_TX_BUFFER_SIZE

```
#define UART_TX_BUFFER_SIZE 256
```

Size of the circular transmit buffer, must be power of 2.

You may need to adapt this constant to your target and your application by adding CDEFS += -DUART_TX_↩ BUFFER_SIZE=nn to your Makefile.

4.4.3 Function Documentation

4.4.3.1 uart1_getc()

```
unsigned int uart1_getc (
            void  )
```

Get received byte of USART1 from ringbuffer. (only available on selected ATmega)

See also

uart_getc

4.4.3.2 uart1_init()

```
void uart1_init (
            unsigned int baudrate )
```

Initialize USART1 (only available on selected ATmegas)

See also

uart_init

4.4.3.3 uart1_putc()

```
void uart1_putc (
            unsigned char data )
```

Put byte to ringbuffer for transmitting via USART1 (only available on selected ATmega)

See also

uart_putc

4.4.3.4 uart1_puts()

```
void uart1_puts (
            const char * s )
```

Put string to ringbuffer for transmitting via USART1 (only available on selected ATmega)

See also

uart_puts

4.4.3.5 uart1_puts_p()

```
void uart1_puts_p (
             const char * s )
```

Put string from program memory to ringbuffer for transmitting via USART1 (only available on selected ATmega)

See also

uart_puts_p

4.4.3.6 uart_getc()

```
unsigned int uart_getc (
             void  )
```

Get received byte from ringbuffer.

Returns in the lower byte the received character and in the higher byte the last receive error. UART_NO_DATA is returned when no data is available.

Returns

lower byte: received byte from ringbuffer

higher byte: last receive status

- **0** successfully received data from UART
- **UART_NO_DATA**

  no receive data available
- **UART_BUFFER_OVERFLOW**

  Receive ringbuffer overflow. We are not reading the receive buffer fast enough, one or more received character have been dropped
- **UART_OVERRUN_ERROR**

  Overrun condition by UART. A character already present in the UART UDR register was not read by the interrupt handler before the next character arrived, one or more received characters have been dropped.
- **UART_FRAME_ERROR**

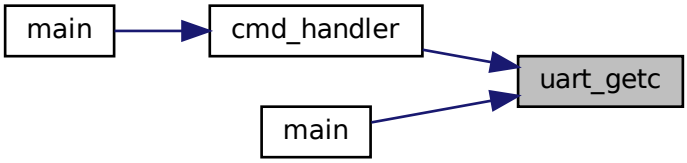
  Framing Error by UART

```
491 {
492     unsigned char tmptail;
493     unsigned char data;
494     unsigned char lastRxError;
495 
496 
497     if ( UART_RxHead == UART_RxTail ) {
498         return UART_NO_DATA;   /* no data available */
499     }
500 
501     /* calculate buffer index */
502     tmptail = (UART_RxTail + 1) & UART_RX_BUFFER_MASK;
503 
504     /* get data from receive buffer */
505     data = UART_RxBuf[tmptail];
```

```
506     lastRxError = UART_LastRxError;
507
508     /* store buffer index */
509     UART_RxTail = tmptail;
510
511     UART_LastRxError = 0;
512     return (lastRxError « 8) + data;
513
514 }/* uart_getc */
```

Referenced by cmd_handler(), and main().

Here is the caller graph for this function:



4.4.3.7 uart_init()

```
void uart_init (
              unsigned int baudrate )
```

Initialize UART and set baudrate.

Parameters

baudrate	Specify baudrate using macro UART_BAUD_SELECT()

Returns

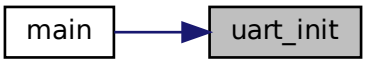
none

```
434 {
435     UART_TxHead = 0;
436     UART_TxTail = 0;
437     UART_RxHead = 0;
438     UART_RxTail = 0;
439
440 #ifdef UART_TEST
441 #ifndef UART0_BIT_U2X
442 #warning "UART0_BIT_U2X not defined"
443 #endif
444 #ifndef UART0_UBRRH
445 #warning "UART0_UBRRH not defined"
446 #endif
447 #ifndef UART0_CONTROLC
448 #warning "UART0_CONTROLC not defined"
449 #endif
450 #if defined(URSEL) || defined(URSEL0)
451 #ifndef UART0_BIT_URSEL
452 #warning "UART0_BIT_URSEL not defined"
```

```
453 #endif
454 #endif
455 #endif
456
457     /* Set baud rate */
458     if ( baudrate & 0x8000 )
459     {
460 #if UART0_BIT_U2X
461         UART0_STATUS = (1<<UART0_BIT_U2X);  //Enable 2x speed
462 #endif
463     }
464 #if defined(UART0_UBRRH)
465     UART0_UBRRH = (unsigned char)((baudrate>>8)&0x80) ;
466 #endif
467     UART0_UBRRL = (unsigned char) (baudrate&0x00FF);
468
469     /* Enable USART receiver and transmitter and receive complete interrupt */
470     UART0_CONTROL = _BV(UART0_BIT_RXCIE)|(1<<UART0_BIT_RXEN)|(1<<UART0_BIT_TXEN);
471
472     /* Set frame format:  asynchronous, 8data, no parity, 1stop bit */
473 #ifdef UART0_CONTROLC
474 #ifdef UART0_BIT_URSEL
475     UART0_CONTROLC = (1<<UART0_BIT_URSEL)|(1<<UART0_BIT_UCSZ1)|(1<<UART0_BIT_UCSZ0);
476 #else
477     UART0_CONTROLC = (1<<UART0_BIT_UCSZ1)|(1<<UART0_BIT_UCSZ0);
478 #endif
479 #endif
480
481 }/* uart_init */
```

Referenced by main().

Here is the caller graph for this function:



4.4.3.8 uart_putc()

```
void uart_putc (
            unsigned char data )
```

Put byte to ringbuffer for transmitting via UART.

Parameters

data	byte to be transmitted

Returns

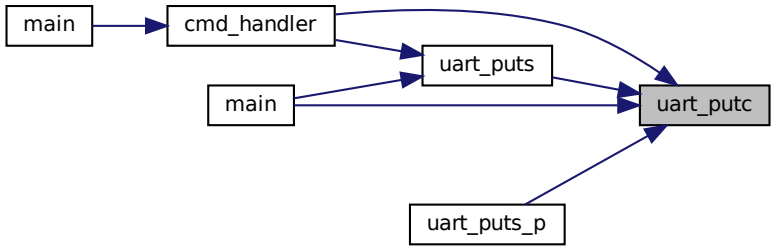
none

```
524 {
525     unsigned char tmphead;
526
```

```
527
528     tmphead  = (UART_TxHead + 1) & UART_TX_BUFFER_MASK;
529
530     while ( tmphead == UART_TxTail ){
531         ;/* wait for free space in buffer */
532     }
533
534     UART_TxBuf[tmphead] = data;
535     UART_TxHead = tmphead;
536
537     /* enable UDRE interrupt */
538     UART0_CONTROL    |= _BV(UART0_UDRIE);
539
540 }/* uart_putc */
```

Referenced by cmd_handler(), main(), uart_puts(), and uart_puts_p().

Here is the caller graph for this function:



4.4.3.9 uart_puts()

```
void uart_puts (
             const char * s )
```

Put string to ringbuffer for transmitting via UART.

The string is buffered by the uart library in a circular buffer and one character at a time is transmitted to the UART using interrupts. Blocks if it can not write the whole string into the circular buffer.

Parameters

s	string to be transmitted

Returns

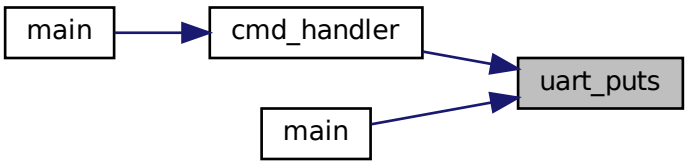
none

```
550 {
551     while (*s)
552       uart_putc(*s++);
553
554 }/* uart_puts */
```

Referenced by cmd_handler(), and main().

Here is the call graph for this function:

Here is the caller graph for this function:



4.4.3.10 uart_puts_p()

```
void uart_puts_p (
            const char * s )
```

Put string from program memory to ringbuffer for transmitting via UART.

The string is buffered by the uart library in a circular buffer and one character at a time is transmitted to the UART using interrupts. Blocks if it can not write the whole string into the circular buffer.

Parameters

s	program memory string to be transmitted

Returns

none

See also

uart_puts_P

```
564 {
565     register char c;
566 
567     while ( (c = pgm_read_byte(progmem_s++)) )
568       uart_putc(c);
569 
570 }/* uart_puts_p */
```

Here is the call graph for this function:

Chapter 5

Class Documentation

5.1 dataset_t Struct Reference

`#include <dataset.h>`

Public Attributes

- uint32_t time
- uint8_t temp
- uint8_t hum
- uint8_t moist

5.1.1 Detailed Description

Structure used for saving of measurements and reading them

5.1.2 Member Data Documentation

5.1.2.1 hum

`uint8_t dataset_t::hum`

5.1.2.2 moist

`uint8_t dataset_t::moist`

5.1.2.3 temp

```
uint8_t dataset_t::temp
```

5.1.2.4 time

```
uint32_t dataset_t::time
```

Referenced by ISR(), and main().

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

- lib/dataset/dataset.h

5.2 servo_t Struct Reference

```
#include <servo.h>
```

Public Attributes

- volatile uint8_t * reg
- uint8_t pin
- uint8_t value

5.2.1 Detailed Description

5.2.1.1 @struct servo_t

5.2.1.2 @var servo::reg

5.2.1.3 @var servo::pin

5.2.1.4 @var servo::value

5.2.2 Member Data Documentation

5.2.2.1 pin

```
uint8_t servo_t::pin
```

Referenced by servo_50us_interrupt_handler(), and servo_init().

5.2.2.2 reg

```
volatile uint8_t* servo_t::reg
```

Referenced by servo_50us_interrupt_handler(), and servo_init().

5.2.2.3 value

```
uint8_t servo_t::value
```

Referenced by servo_50us_interrupt_handler(), servo_set_value(), and watering_handler().

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

- lib/servo/servo.h

5.3 storage_t Struct Reference

```
#include <storage.h>
```

Public Attributes

- uint16_t buffer_start

5.3.1 Member Data Documentation

5.3.1.1 buffer_start

```
uint16_t storage_t::buffer_start
```

Referenced by storage_init(), storage_read(), and storage_write().

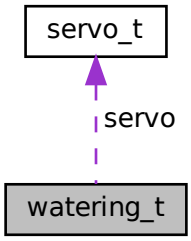
The documentation for this struct was generated from the following file:

- lib/storage/storage.h

5.4 watering_t Struct Reference

`#include <watering.h>`

Collaboration diagram for watering_t:



Public Attributes

- servo_t * servo
- uint16_t min
- uint16_t max

5.4.1 Detailed Description

5.4.1.1 @struct watering_t

5.4.1.2 @var watering::servo

5.4.1.3 @var watering::min

5.4.1.4 @var watering::max

5.4.2 Member Data Documentation

5.4.2.1 max

`uint16_t watering_t::max`

Referenced by cmd_handler(), watering_handler(), and watering_set_limit().

5.4.2.2 min

```
uint16_t watering_t::min
```

Referenced by cmd_handler(), watering_handler(), and watering_set_limit().

5.4.2.3 servo

```
servo_t* watering_t::servo
```

Referenced by watering_handler(), and watering_init().

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

- lib/watering/watering.h

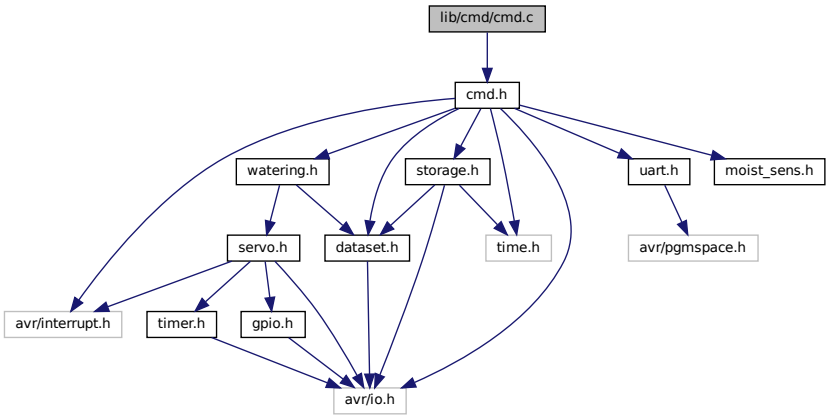
Chapter 6

File Documentation

6.1 lib/cmd/cmd.c File Reference

```
#include "cmd.h"
```

Include dependency graph for cmd.c:



Functions

- void cmd_handler (dataset_t *data, watering_t *watering, storage_t *storage)

6.1.1 Function Documentation

6.1.1.1 cmd_handler()

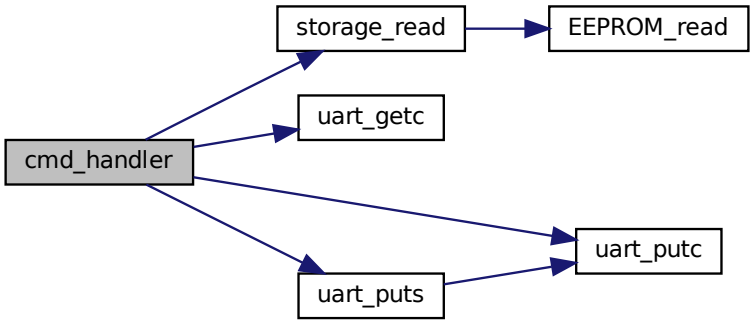
```
void cmd_handler (
            dataset_t * data,
            watering_t * watering,
            storage_t * storage )
{
    // By typing different letters into command line this finction shows help menu or data form sensors
     uint8_t value, tmp;          // Selected letter is saved into this variable
     //uint16_t moisture = 50;      // Soil Moisture is saved here
     //uint8_t humidity = 60;       // Air humidity is saved here
     //uint8_t temperature = 25;    // Air temperature is saved here
     char string[8];  // String for converted numbers by itoa()

     struct tm* local;
     time_t t;

     int limit_tmp, data_n = 0;
     dataset_t mydata;

     // Get the localtime
     t = data->time;
     local = localtime(&t);


     value = uart_getc();
     if (value > 0) {  // Data available from UART

         switch (value) {
             case '?':        // By typing '?'  into command line the Help menu is shown
                 uart_puts("\nHelp:  \n"
                           "?  - Show help\n"
                           "a - Show every information\n"
                           "A N - Show every information for N last minutes\n"
                           "m - Show soil moisture\n"
                           "t - Show air temperature\n"
                           "h - Show air humidity\n"
                           "c - Show current time and date\n"
                           "C UNIT_TIME - Set current unix time\n"
                           "l - Show moisture limits"
                           "L MIN MAX - Set moister limits\n"
                 );
                 break;

             case 'm':      // By typing 'm' program will give you current soil moisture
                 uart_puts("\nSoil moisture:  ");
                 //moisture = get_moist();
                 itoa(data->moist, string, 10);
                 uart_puts(string);
                 uart_puts("/255\n");
                 break;

             case 't':      // By typing 't' program will give you current air temperature
                 uart_puts("\nAir temperature:  ");
                 // temperature = get_temperature();
                 itoa(data->temp, string, 10);
                 uart_puts(string);
                 uart_puts("°C\n");
                 break;

             case 'h':      // By typing 'h' program will give you current air humidity
                 uart_puts("\nAir humidity:  ");
                 // humidity = get_humidity();
                 itoa(data->hum, string, 10);
                 uart_puts(string);
                 uart_puts("%\n");
                 break;

             case 'L':
                 limit_tmp = 0;
                 while((tmp = uart_getc()) != '\n' && tmp != ' ') // TEST THIS
                 {
                     if(tmp == UART_NO_DATA || tmp < '0' || tmp > '9') continue;
                     uart_putc(tmp);
                     limit_tmp = (limit_tmp * 10) + tmp - '0';
                 }
                 uart_puts(" - ");
                 watering->min = limit_tmp;
                 limit_tmp = 0;
                 while((tmp = uart_getc()) != '\n')
                 {
                     if(tmp == UART_NO_DATA || tmp < '0' || tmp > '9') continue;
                     uart_putc(tmp);
                     limit_tmp = (limit_tmp * 10) + tmp - '0';
```

```
86                  }
87                  watering->max = limit_tmp;
88
89                  uart_puts("\nLimits were set");
90
91              case 'l':       // By typing 's' program will give you current moisture limis
92                  uart_puts("\nMoisture limits:  from ");
93                  itoa(watering->min, string, 10);
94                  uart_puts(string);
95                  uart_puts(" to ");
96                  itoa(watering->max, string, 10);
97                  uart_puts(string);
98                  uart_puts("\n");
99                  break;
100
101             case 'C':
102                 t = 0;
103                 while((tmp = uart_getc()) != '\n')
104                 {
105                     if(tmp == UART_NO_DATA || tmp < '0' || tmp > '9') continue;
106                     uart_putc(tmp);
107                     t = (t * 10) + tmp - '0';
108                 }
109                 data->time = t - AVRTIME_TO_UNIXTIME;
110                 uart_puts("Time was set");
111
112             case 'c':         // By typing 'c' program will give you current time
113                 uart_puts("\nCurrent time and date:  ");
114                 uart_puts(asctime(local));
115                 uart_puts("\n");
116                 break;
117
118             case 'A':
119                 data_n = 0;
120                 while((tmp = uart_getc()) != '\n')
121                 {
122                     if(tmp == UART_NO_DATA || tmp < '0' || tmp > '9') continue;
123                     uart_putc(tmp);
124                     data_n = (data_n * 10) + tmp - '0';
125                 }
126                 data = &mydata;
127
128                 storage_read(storage, data, data_n+1);
129                 t = data->time;
130                 local = localtime(&t);
131
132                 //data_n--;
133
134             case 'a':         // By typing 'a' program will give every current information
135                 uart_puts("\nDate\tTemp [C]\tHum [%]\tMoist [1/255]\n");
136                 while(1)
137                 {
138                     // Time
139                     uart_puts(asctime(local));
140                     uart_puts("\t");
141
142                     // Temperature
143                     itoa(data->temp, string, 10);
144                     uart_puts(string);
145                     uart_puts("\t");
146
147                     // Humidity
148                     itoa(data->hum, string, 10);
149                     uart_puts(string);
150                     uart_puts("\t");
151
152                     // Moisture
153                     itoa(data->moist, string, 10);
154                     uart_puts(string);
155                     uart_puts("\n");
156
157                     if(data_n > 1)
158                     {
159
160                         data_n--;
161                         storage_read(storage, data, data_n);
162                         t = data->time;
163                         local = localtime(&t);
164                     }
165                     else break;
166
167                 }
168                 break;
169
170
171             default:        // When you type different letter the program will try to help you
172                 uart_puts("\nWrong letter was typed.  Type \x27?\x27 for help\n");
```

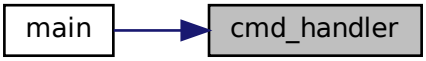
```
173                 break;
174         }
175
176         // uart_putc('\n');
177     }
178
179 }
```

Referenced by main().

Here is the call graph for this function:



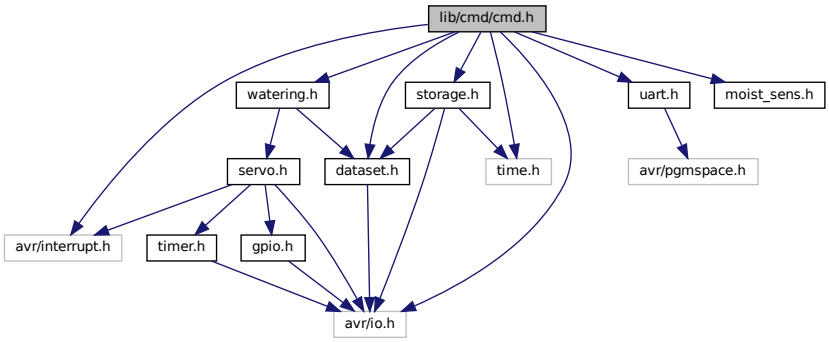
Here is the caller graph for this function:



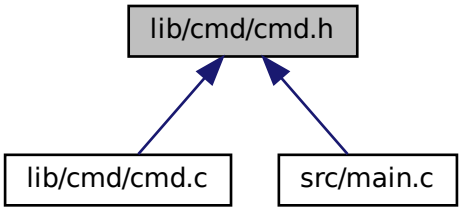
6.2 lib/cmd/cmd.h File Reference

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <time.h>
#include <uart.h>
#include <moist_sens.h>
#include <dataset.h>
#include <watering.h>
```

`#include <storage.h>`
Include dependency graph for cmd.h:



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



Functions

- void cmd_handler (dataset_t *data, watering_t *watering, storage_t *storage)

6.2.1 Function Documentation

6.2.1.1 cmd_handler()

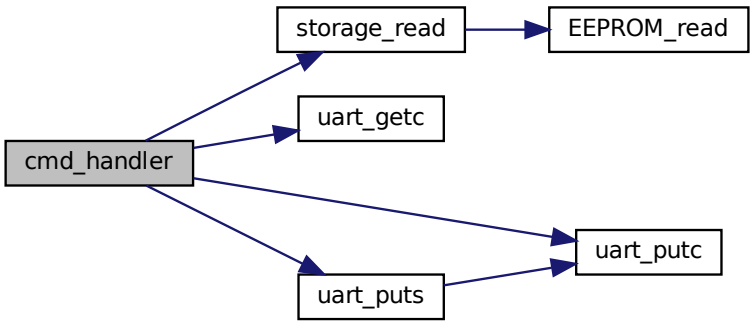
```
void cmd_handler (
              dataset_t * data,
              watering_t * watering,
              storage_t * storage )
8 {
9     // By typing different letters into command line this finction shows help menu or data form sensors
10     uint8_t value, tmp;          // Selected letter is saved into this variable
11     //uint16_t moisture = 50;      // Soil Moisture is saved here
```

```
12      //uint8_t humidity = 60;       // Air humidity is saved here
13      //uint8_t temperature = 25;    // Air temperature is saved here
14      char string[8];  // String for converted numbers by itoa()
15
16      struct tm* local;
17      time_t t;
18
19      int limit_tmp, data_n = 0;
20      dataset_t mydata;
21
22      // Get the localtime
23      t = data->time;
24      local = localtime(&t);
25
26
27      value = uart_getc();
28      if (value > 0) {  // Data available from UART
29
30          switch (value) {
31              case '?':        // By typing '?'  into command line the Help menu is shown
32                  uart_puts("\nHelp:  \n"
33                            "?  - Show help\n"
34                            "a - Show every information\n"
35                            "A N - Show every information for N last minutes\n"
36                            "m - Show soil moisture\n"
37                            "t - Show air temperature\n"
38                            "h - Show air humidity\n"
39                            "c - Show current time and date\n"
40                            "C UNIT_TIME - Set current unix time\n"
41                            "l - Show moisture limits"
42                            "L MIN MAX - Set moister limits\n"
43                  );
44                  break;
45
46              case 'm':      // By typing 'm' program will give you current soil moisture
47                  uart_puts("\nSoil moisture:  ");
48                  //moisture = get_moist();
49                  itoa(data->moist, string, 10);
50                  uart_puts(string);
51                  uart_puts("/255\n");
52                  break;
53
54              case 't':      // By typing 't' program will give you current air temperature
55                  uart_puts("\nAir temperature:  ");
56                  // temperature = get_temperature();
57                  itoa(data->temp, string, 10);
58                  uart_puts(string);
59                  uart_puts("°C\n");
60                  break;
61
62              case 'h':      // By typing 'h' program will give you current air humidity
63                  uart_puts("\nAir humidity:  ");
64                  // humidity = get_humidity();
65                  itoa(data->hum, string, 10);
66                  uart_puts(string);
67                  uart_puts("%\n");
68                  break;
69
70              case 'L':
71                  limit_tmp = 0;
72                  while((tmp = uart_getc()) != '\n' && tmp != ' ') // TEST THIS
73                  {
74                      if(tmp == UART_NO_DATA || tmp < '0' || tmp > '9') continue;
75                      uart_putc(tmp);
76                      limit_tmp = (limit_tmp * 10) + tmp - '0';
77                  }
78                  uart_puts(" - ");
79                  watering->min = limit_tmp;
80                  limit_tmp = 0;
81                  while((tmp = uart_getc()) != '\n')
82                  {
83                      if(tmp == UART_NO_DATA || tmp < '0' || tmp > '9') continue;
84                      uart_putc(tmp);
85                      limit_tmp = (limit_tmp * 10) + tmp - '0';
86                  }
87                  watering->max = limit_tmp;
88
89                  uart_puts("\nLimits were set");
90
91              case 'l':      // By typing 's' program will give you current moisture limis
92                  uart_puts("\nMoisture limits:  from ");
93                  itoa(watering->min, string, 10);
94                  uart_puts(string);
95                  uart_puts(" to ");
96                  itoa(watering->max, string, 10);
97                  uart_puts(string);
98                  uart_puts("\n");
```

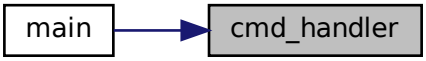
```
99                  break;
100
101             case 'C':
102                 t = 0;
103                 while((tmp = uart_getc()) != '\n')
104                 {
105                     if(tmp == UART_NO_DATA || tmp < '0' || tmp > '9') continue;
106                     uart_putc(tmp);
107                     t = (t * 10) + tmp - '0';
108                 }
109                 data->time = t - AVRTIME_TO_UNIXTIME;
110                 uart_puts("Time was set");
111
112             case 'c':        // By typing 'c' program will give you current time
113                 uart_puts("\nCurrent time and date:  ");
114                 uart_puts(asctime(local));
115                 uart_puts("\n");
116                 break;
117
118             case 'A':
119                 data_n = 0;
120                 while((tmp = uart_getc()) != '\n')
121                 {
122                     if(tmp == UART_NO_DATA || tmp < '0' || tmp > '9') continue;
123                     uart_putc(tmp);
124                     data_n = (data_n * 10) + tmp - '0';
125                 }
126                 data = &mydata;
127
128                 storage_read(storage, data, data_n+1);
129                 t = data->time;
130                 local = localtime(&t);
131
132                 //data_n--;
133
134             case 'a':        // By typing 'a' program will give every current information
135                 uart_puts("\nDate\tTemp [C]\tHum [%]\tMoist [1/255]\n");
136                 while(1)
137                 {
138                     // Time
139                     uart_puts(asctime(local));
140                     uart_puts("\t");
141
142                     // Temperature
143                     itoa(data->temp, string, 10);
144                     uart_puts(string);
145                     uart_puts("\t");
146
147                     // Humidity
148                     itoa(data->hum, string, 10);
149                     uart_puts(string);
150                     uart_puts("\t");
151
152                     // Moisture
153                     itoa(data->moist, string, 10);
154                     uart_puts(string);
155                     uart_puts("\n");
156
157                     if(data_n > 1)
158                     {
159
160                         data_n--;
161                         storage_read(storage, data, data_n);
162                         t = data->time;
163                         local = localtime(&t);
164                     }
165                     else break;
166
167                 }
168                 break;
169
170
171             default:        // When you type different letter the program will try to help you
172                 uart_puts("\nWrong letter was typed.  Type \x27?\x27 for help\n");
173                 break;
174         }
175
176         // uart_putc('\n');
177     }
178
179 }
```

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



6.3 cmd.h

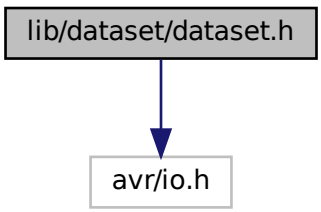
Go to the documentation of this file.

```
1 #ifndef CMD_H
2 #define CMD_H
3
4 #include <avr/io.h>
5 #include <avr/interrupt.h>
6 #include <time.h>
7 #include <uart.h>
8 #include <moist_sens.h>
9 #include <dataset.h>
10 #include <watering.h>
11 #include <storage.h>
12
13 /************************************************
14 *  @brief  Enables communication
15 *          between user and program via command line
16 *  @param  data Actual measured data
17 *  @param  watering Watering setup
18 *  @param  storage Storage setup
19 *  @return None
20 ************************************************/
21
22 void cmd_handler(dataset_t *data, watering_t *watering, storage_t *storage);
23
24
25 #endif
26
```

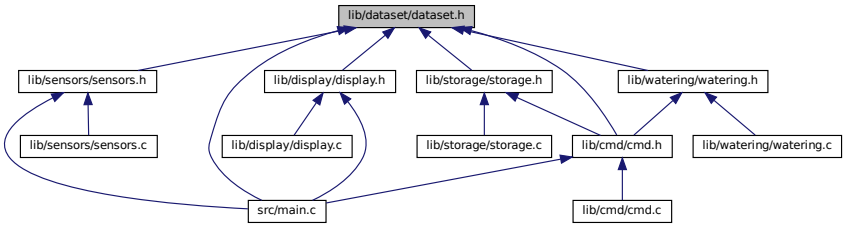
6.4 lib/dataset/dataset.h File Reference

```
#include <avr/io.h>
```

Include dependency graph for dataset.h:



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



Classes

- struct dataset_t

Macros

- #define AVRTIME_TO_UNIXTIME 946681200

6.4.1 Macro Definition Documentation

6.4.1.1 AVRTIME_TO_UNIXTIME

```
#define AVRTIME_TO_UNIXTIME 946681200
```

6.5 dataset.h

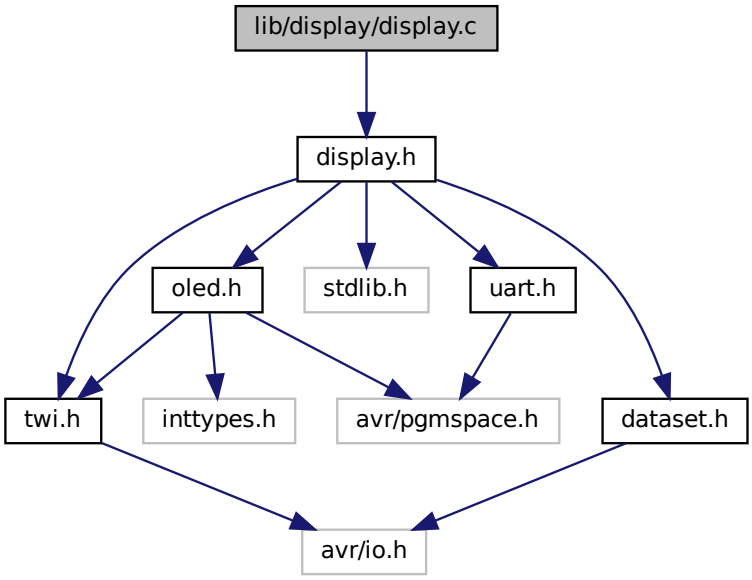
Go to the documentation of this file.

```
1 #ifndef DATASET_H
2 #define DATASET_H
3
4 #include <avr/io.h>
5
6 #define AVRTIME_TO_UNIXTIME 946681200
7
22 typedef struct  {
23    uint32_t time;
24    uint8_t temp;
25    uint8_t hum;
26    uint8_t moist;
27 } dataset_t;
28
29 #endif
30
```

6.6 lib/display/display.c File Reference

`#include "display.h"`

Include dependency graph for display.c:



Functions

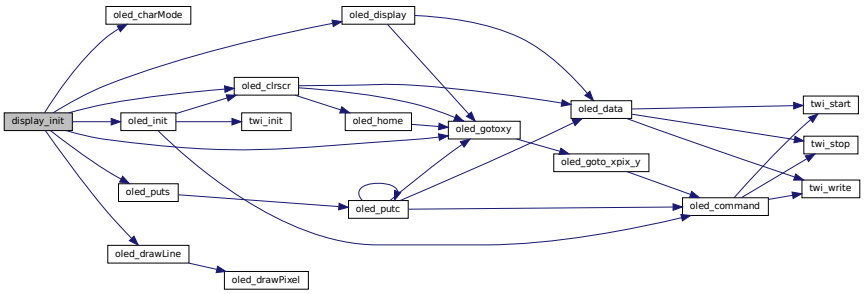
- void display_init ()
- void display_show_data (dataset_t *data)

6.6.1 Function Documentation

6.6.1.1 display_init()

```
void display_init ( )
4 {
5     oled_init(OLED_DISP_ON);
6     oled_clrscr();
7
8     oled_charMode(DOUBLESIZE);
9     // oled_puts("OLED disp.");
10
11     oled_charMode(NORMALSIZE);
12
13     oled_gotoxy(0, 1);
14     oled_puts("Podminky v kvetinaci");
15
16     oled_gotoxy(0, 2);
17     // oled_drawLine(x1, y1, x2, y2, color)
18     oled_drawLine(0, 25, 120, 25, WHITE);
19
20     // Copy buffer to display RAM
21     oled_display();
22 }
```

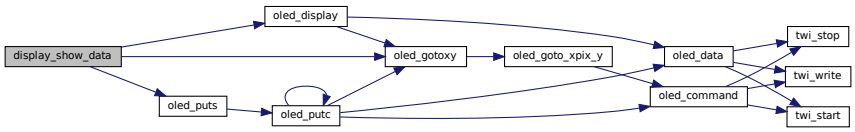
Here is the call graph for this function:



6.6.1.2 display_show_data()

```
void display_show_data (
             dataset_t * data )
25 {
26     char tmp_str[20];
27
28     itoa(data->temp, tmp_str, 10);
29     oled_gotoxy(0, 4);
30     oled_puts("Teplota vzduchu\t");
31     oled_puts(tmp_str);
32     oled_puts(" °C");
33     oled_display();
34
35     itoa(data->hum, tmp_str, 10);
36
37     oled_gotoxy(0, 5);
38     oled_puts("Vlhkost vzduchu\t");
39     oled_puts(tmp_str);
40     oled_puts(" %");
41
42     for(long i=0; i < 5000; i++) asm("NOP"); // around 1ms delay
43
44     oled_display();
45 }
```

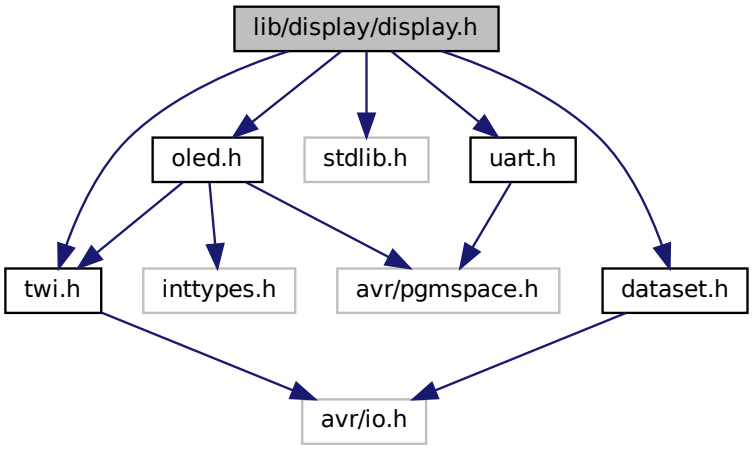
Here is the call graph for this function:



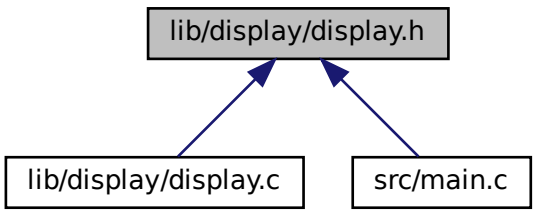
6.7 lib/display/display.h File Reference

```
#include <twi.h>
#include <uart.h>
#include <stdlib.h>
#include <oled.h>
#include <dataset.h>
```

Include dependency graph for display.h:



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



Functions

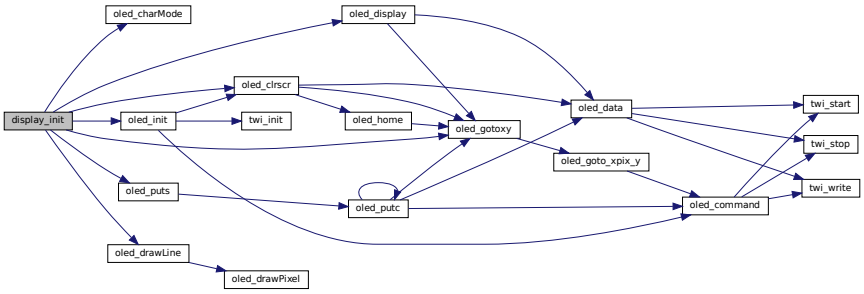
- void display_init ()
- void display_show_data (dataset_t *data)

6.7.1 Function Documentation

6.7.1.1 display_init()

```
void display_init ( )
4 {
5     oled_init(OLED_DISP_ON);
6     oled_clrscr();
7
8     oled_charMode(DOUBLESIZE);
9     // oled_puts("OLED disp.");
10
11     oled_charMode(NORMALSIZE);
12
13     oled_gotoxy(0, 1);
14     oled_puts("Podminky v kvetinaci");
15
16     oled_gotoxy(0, 2);
17     // oled_drawLine(x1, y1, x2, y2, color)
18     oled_drawLine(0, 25, 120, 25, WHITE);
19
20     // Copy buffer to display RAM
21     oled_display();
22 }
```

Here is the call graph for this function:

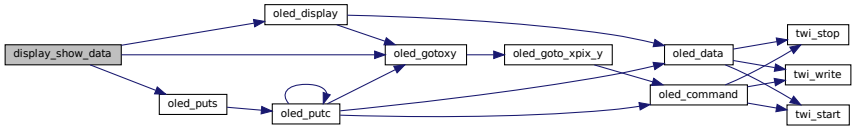


6.7.1.2 display_show_data()

```
void display_show_data (
            dataset_t * data )
25 {
26     char tmp_str[20];
27
28     itoa(data->temp, tmp_str, 10);
29     oled_gotoxy(0, 4);
30     oled_puts("Teplota vzduchu\t");
31     oled_puts(tmp_str);
32     oled_puts(" °C");
```

```
33     oled_display();
34
35     itoa(data->hum, tmp_str, 10);
36
37     oled_gotoxy(0, 5);
38     oled_puts("Vlhkost vzduchu\t");
39     oled_puts(tmp_str);
40     oled_puts(" %");
41
42     for(long i=0; i < 5000; i++) asm("NOP"); // around 1ms delay
43
44     oled_display();
45 }
```

Here is the call graph for this function:



6.8 display.h

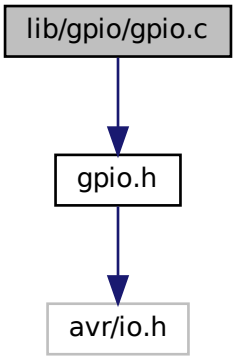
Go to the documentation of this file.

```
1 #ifndef DISPLAY_H
2 #define DISPLAY_H
3
4 #include <twi.h>            // I2C/TWI library for AVR-GCC
5 #include <uart.h>           // Peter Fleury's UART library
6 #include <stdlib.h>         // C library.  Needed for number conversions
7 #include <oled.h>           // OLED
8 #include <dataset.h>
9
10
11 /*****************************************************
12 *  @brief  Inicialization of diplay
13 *  @return None
14 *****************************************************/
15
16 void display_init();
17
18
19 /*****************************************************
20 *  @brief  It dispalys measured data
21 *  @param  data Actual measured data
22 *  @return None
23 *****************************************************/
24
25 void display_show_data(dataset_t *data);
26
27 #endif
28
```

6.9 lib/gpio/gpio.c File Reference

```
#include <gpio.h>
```

Include dependency graph for gpio.c:



Functions

- void GPIO_mode_output (volatile uint8_t *reg, uint8_t pin)

  Configure one output pin.
- void GPIO_mode_input_pullup (volatile uint8_t *reg, uint8_t pin)

  Configure one input pin and enable pull-up.
- void GPIO_write_low (volatile uint8_t *reg, uint8_t pin)

  Write one pin to low value.
- void GPIO_write_high (volatile uint8_t *reg, uint8_t pin)

  Write one pin to high value.
- void GPIO_write (volatile uint8_t *reg, uint8_t pin, uint8_t value)

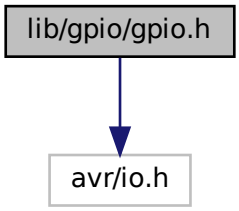
  Write one pin to specific value.
- uint8_t GPIO_read (volatile uint8_t *reg, uint8_t pin)

  Read a value from input pin.

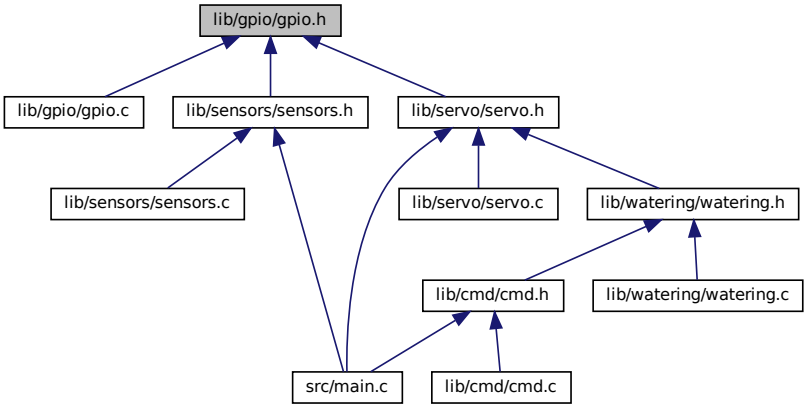
6.10 lib/gpio/gpio.h File Reference

```
#include <avr/io.h>
```

Include dependency graph for gpio.h:



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



Functions

- void GPIO_mode_output (volatile uint8_t ∗reg, uint8_t pin)

  Configure one output pin.
- void GPIO_mode_input_pullup (volatile uint8_t ∗reg, uint8_t pin)

  Configure one input pin and enable pull-up.
- void GPIO_write_low (volatile uint8_t ∗reg, uint8_t pin)

  Write one pin to low value.
- void GPIO_write_high (volatile uint8_t ∗reg, uint8_t pin)

  Write one pin to high value.
- void GPIO_write (volatile uint8_t ∗reg, uint8_t pin, uint8_t value)

  Write one pin to specific value.
- uint8_t GPIO_read (volatile uint8_t ∗reg, uint8_t pin)

  Read a value from input pin.

6.11 gpio.h

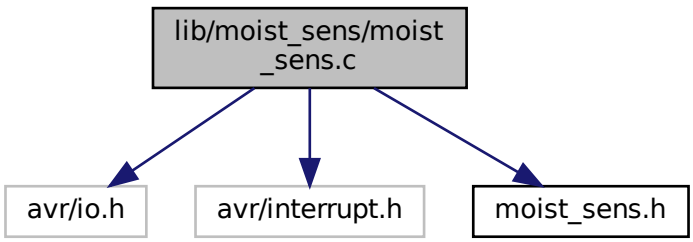
Go to the documentation of this file.

```
1 #ifndef GPIO_H
2 # define GPIO_H
3
4 /*****************************************************************************
5 *
6 * GPIO library for AVR-GCC.
7 *
8 * ATmega328P (Arduino Uno), 16 MHz, PlatformIO
9 *
10 * Copyright (c) 2019 Tomas Fryza
11 * Dept.  of Radio Electronics, Brno University of Technology, Czechia
12 * This work is licensed under the terms of the MIT license.
13 *
14 *****************************************************************************/
15
35 /* Includes ------------------------------------------------------------*/
36 #include <avr/io.h>
37
38
39 /* Function prototypes -------------------------------------------------*/
46 void GPIO_mode_output(volatile uint8_t *reg, uint8_t pin);
47
48
55 void GPIO_mode_input_pullup(volatile uint8_t *reg, uint8_t pin);
56
57
64 void GPIO_write_low(volatile uint8_t *reg, uint8_t pin);
65
66
73 void GPIO_write_high(volatile uint8_t *reg, uint8_t pin);
74
82 void GPIO_write(volatile uint8_t *reg, uint8_t pin, uint8_t value);
83
84
91 uint8_t GPIO_read(volatile uint8_t *reg, uint8_t pin);
92
93
94 /* GPIO_mode_input_nopull */
95
96
97 /* GPIO_toggle */
98
99
102 #endif
```

6.12 lib/moist_sens/moist_sens.c File Reference

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include "moist_sens.h"
```

Include dependency graph for moist_sens.c:



Functions

- void moist_sens_init (void)
- uint16_t get_moist (void)
- ISR (ADC_vect)

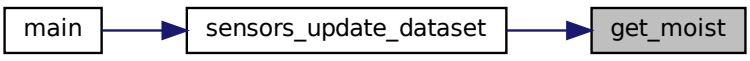
6.12.1 Function Documentation

6.12.1.1 get_moist()

```
uint16_t get_moist (
              void  )
39 {
40     // Start ADC conversion
41     ADCSRA = ADCSRA | (1«ADSC);
42
43     uint16_t moisture;
44     uint16_t zero_moist = 800;
45     uint16_t max_moist = 680;
46     uint16_t moist_constant;
47
48     while(ADCSRA & (1«ADSC));
49
50     moisture = ADC;
51     /*moisture = moisture - max_moist;
52 moist_constant = zero_moist - max_moist;
53 moisture = moisture*100;
54 moisture = moisture/moist_constant;
55 moisture = 100 - moisture;*/
56
57     return 255 - (moisture/4);
58
59
60 }
```

Referenced by sensors_update_dataset().

Here is the caller graph for this function:



6.12.1.2 ISR()

```
ISR (
              ADC_vect  )
63 {
64     asm("NOP");
65 }
```

6.12.1.3 moist_sens_init()

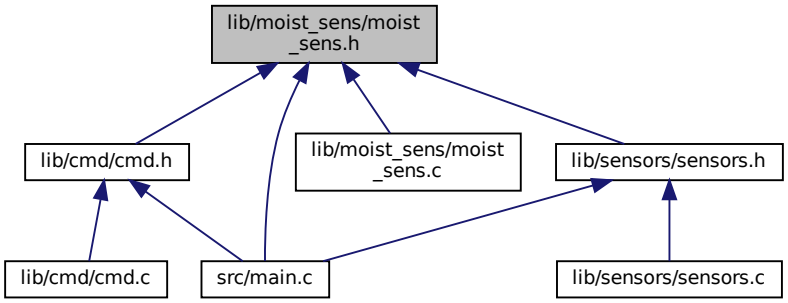
```
void moist_sens_init (
             void  )
14                          {
15
16     // Configure Analog-to-Digital Convertion unit
17     // Select ADC voltage reference to "AVcc with external capacitor at AREF pin"
18     ADMUX = ADMUX | (1«REFS0);
19     // Select input channel ADC0 (voltage divider pin)
20     ADMUX = ADMUX & ~(1«MUX3 | 1«MUX2 | 1«MUX1 | 1«MUX0);
21     // Enable ADC module
22     ADCSRA = ADCSRA | (1«ADEN);
23     // Enable conversion complete interrupt
24     ADCSRA = ADCSRA | (1«ADIE);
25     // Set clock prescaler to 128
26     ADCSRA = ADCSRA | (1«ADPS2 | 1«ADPS1 | 1«ADPS0);
27 }
```

Referenced by sensors_init().

Here is the caller graph for this function:

6.13 lib/moist_sens/moist_sens.h File Reference

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



Functions

- void moist_sens_init (void)
- uint16_t get_moist (void)

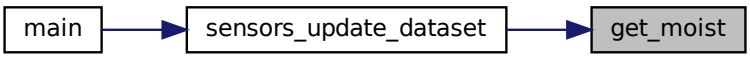
6.13.1 Function Documentation

6.13.1.1 get_moist()

```
uint16_t get_moist (
              void  )
39 {
40     // Start ADC conversion
41     ADCSRA = ADCSRA | (1«ADSC);
42
43     uint16_t moisture;
44     uint16_t zero_moist = 800;
45     uint16_t max_moist = 680;
46     uint16_t moist_constant;
47
48     while(ADCSRA & (1«ADSC));
49
50     moisture = ADC;
51     /*moisture = moisture - max_moist;
52 moist_constant = zero_moist - max_moist;
53 moisture = moisture*100;
54 moisture = moisture/moist_constant;
55 moisture = 100 - moisture;*/
56
57     return 255 - (moisture/4);
58
59
60 }
```

Referenced by sensors_update_dataset().

Here is the caller graph for this function:



6.13.1.2 moist_sens_init()

```
void moist_sens_init (
              void  )
14                          {
15
16     // Configure Analog-to-Digital Convertion unit
17     // Select ADC voltage reference to "AVcc with external capacitor at AREF pin"
18     ADMUX = ADMUX | (1«REFS0);
19     // Select input channel ADC0 (voltage divider pin)
20     ADMUX = ADMUX & ~(1«MUX3 | 1«MUX2 | 1«MUX1 | 1«MUX0);
21     // Enable ADC module
22     ADCSRA = ADCSRA | (1«ADEN);
23     // Enable conversion complete interrupt
24     ADCSRA = ADCSRA | (1«ADIE);
25     // Set clock prescaler to 128
26     ADCSRA = ADCSRA | (1«ADPS2 | 1«ADPS1 | 1«ADPS0);
27 }
```

Referenced by sensors_init().

Here is the caller graph for this function:

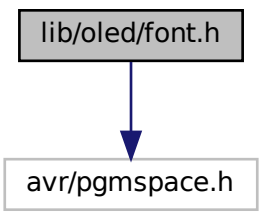
6.14 moist_sens.h

Go to the documentation of this file.

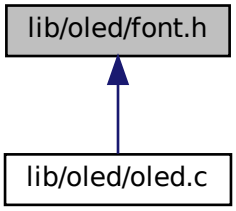
```
1 #ifndef MOIST_SENS_H
2 # define MOIST_SENS_H
3
4 /************************************************
5 *  @brief Configures the ADC and input pin
6 *
7 *  @param  None
8 *  @return None
9 ************************************************/
10 void moist_sens_init(void);
11
12 /************************************************
13 *  @brief Starts the ADC conversion on
14 *         pin PC0 (A0) and returns converted value
15 *
16 *  @param  None
17 *  @return Returns measured moisture in percents
18 ************************************************/
19 uint16_t get_moist(void);
20
21
22
23 #endif
```

6.15 lib/oled/font.h File Reference

`#include <avr/pgmspace.h>`
Include dependency graph for font.h:



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



Variables

- const char ssd1306oled_font[ ][6] PROGMEM

6.15.1 Variable Documentation

6.15.1.1 PROGMEM

```
const char special_char [][2] PROGMEM
```

6.16 font.h

Go to the documentation of this file.

```
1 /*
2 *  font.h
3 *
4 *  Created by Michael Köhler on 13.09.18.
5 *  Copyright 2018 Skie-Systems.  All rights reserved.
6 *
7 */
8 #ifndef _font_h_
9 # define _font_h_
10 # include <avr/pgmspace.h>
11
12 // extern const char ssd1306oled_font[][6] PROGMEM;
13 // extern const char special_char[][2] PROGMEM;
14
15 const char ssd1306oled_font[][6] PROGMEM = {
16     {0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, // sp
17     {0x00, 0x00, 0x00, 0x2f, 0x00, 0x00}, // !
18     {0x00, 0x00, 0x07, 0x00, 0x07, 0x00}, // "
19     {0x00, 0x14, 0x7f, 0x14, 0x7f, 0x14}, // #
20     {0x00, 0x24, 0x2a, 0x7f, 0x2a, 0x12}, // $
21     {0x00, 0x62, 0x64, 0x08, 0x13, 0x23}, // %
22     {0x00, 0x36, 0x49, 0x55, 0x22, 0x50}, // &
23     {0x00, 0x00, 0x05, 0x03, 0x00, 0x00}, // '
24     {0x00, 0x00, 0x1c, 0x22, 0x41, 0x00}, // (
25     {0x00, 0x00, 0x41, 0x22, 0x1c, 0x00}, // )
26     {0x00, 0x14, 0x08, 0x3E, 0x08, 0x14}, // *
27     {0x00, 0x08, 0x08, 0x3E, 0x08, 0x08}, // +
28     {0x00, 0x00, 0x00, 0xA0, 0x60, 0x00}, // ,
```

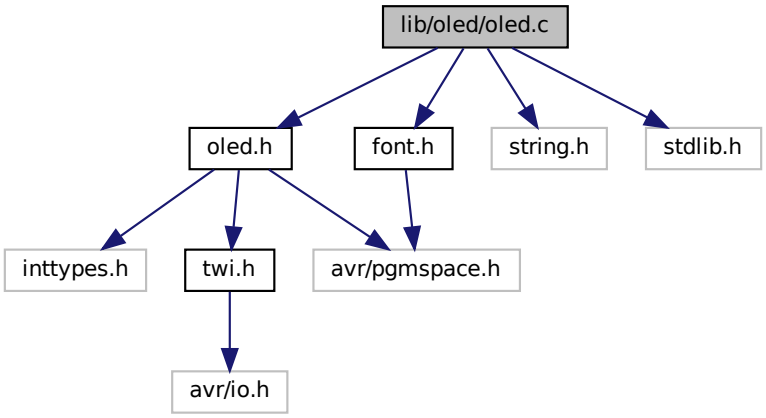
```
29      {0x00, 0x08, 0x08, 0x08, 0x08, 0x08}, // -
30      {0x00, 0x00, 0x60, 0x60, 0x00, 0x00}, // .
31      {0x00, 0x20, 0x10, 0x08, 0x04, 0x02}, // /
32      {0x00, 0x3E, 0x51, 0x49, 0x45, 0x3E}, // 0
33      {0x00, 0x00, 0x42, 0x7F, 0x40, 0x00}, // 1
34      {0x00, 0x42, 0x61, 0x51, 0x49, 0x46}, // 2
35      {0x00, 0x21, 0x41, 0x45, 0x4B, 0x31}, // 3
36      {0x00, 0x18, 0x14, 0x12, 0x7F, 0x10}, // 4
37      {0x00, 0x27, 0x45, 0x45, 0x45, 0x39}, // 5
38      {0x00, 0x3C, 0x4A, 0x49, 0x49, 0x30}, // 6
39      {0x00, 0x01, 0x71, 0x09, 0x05, 0x03}, // 7
40      {0x00, 0x36, 0x49, 0x49, 0x49, 0x36}, // 8
41      {0x00, 0x06, 0x49, 0x49, 0x29, 0x1E}, // 9
42      {0x00, 0x00, 0x36, 0x36, 0x00, 0x00}, // :
43      {0x00, 0x00, 0x56, 0x36, 0x00, 0x00}, // ;
44      {0x00, 0x08, 0x14, 0x22, 0x41, 0x00}, // <
45      {0x00, 0x14, 0x14, 0x14, 0x14, 0x14}, // =
46      {0x00, 0x00, 0x41, 0x22, 0x14, 0x08}, // >
47      {0x00, 0x02, 0x01, 0x51, 0x09, 0x06}, // ?
48      {0x00, 0x32, 0x49, 0x59, 0x51, 0x3E}, // @
49      {0x00, 0x7C, 0x12, 0x11, 0x12, 0x7C}, // A
50      {0x00, 0x7F, 0x49, 0x49, 0x49, 0x36}, // B
51      {0x00, 0x3E, 0x41, 0x41, 0x41, 0x22}, // C
52      {0x00, 0x7F, 0x41, 0x41, 0x22, 0x1C}, // D
53      {0x00, 0x7F, 0x49, 0x49, 0x49, 0x41}, // E
54      {0x00, 0x7F, 0x09, 0x09, 0x09, 0x01}, // F
55      {0x00, 0x3E, 0x41, 0x49, 0x49, 0x7A}, // G
56      {0x00, 0x7F, 0x08, 0x08, 0x08, 0x7F}, // H
57      {0x00, 0x00, 0x41, 0x7F, 0x41, 0x00}, // I
58      {0x00, 0x20, 0x40, 0x41, 0x3F, 0x01}, // J
59      {0x00, 0x7F, 0x08, 0x14, 0x22, 0x41}, // K
60      {0x00, 0x7F, 0x40, 0x40, 0x40, 0x40}, // L
61      {0x00, 0x7F, 0x02, 0x0C, 0x02, 0x7F}, // M
62      {0x00, 0x7F, 0x04, 0x08, 0x10, 0x7F}, // N
63      {0x00, 0x3E, 0x41, 0x41, 0x41, 0x3E}, // O
64      {0x00, 0x7F, 0x09, 0x09, 0x09, 0x06}, // P
65      {0x00, 0x3E, 0x41, 0x51, 0x21, 0x5E}, // Q
66      {0x00, 0x7F, 0x09, 0x19, 0x29, 0x46}, // R
67      {0x00, 0x46, 0x49, 0x49, 0x49, 0x31}, // S
68      {0x00, 0x01, 0x01, 0x7F, 0x01, 0x01}, // T
69      {0x00, 0x3F, 0x40, 0x40, 0x40, 0x3F}, // U
70      {0x00, 0x1F, 0x20, 0x40, 0x20, 0x1F}, // V
71      {0x00, 0x3F, 0x40, 0x38, 0x40, 0x3F}, // W
72      {0x00, 0x63, 0x14, 0x08, 0x14, 0x63}, // X
73      {0x00, 0x07, 0x08, 0x70, 0x08, 0x07}, // Y
74      {0x00, 0x61, 0x51, 0x49, 0x45, 0x43}, // Z
75      {0x00, 0x00, 0x7F, 0x41, 0x41, 0x00}, // [
76      {0x00, 0x55, 0x2A, 0x55, 0x2A, 0x55}, // backslash
77      {0x00, 0x00, 0x41, 0x41, 0x7F, 0x00}, // ]
78      {0x00, 0x04, 0x02, 0x01, 0x02, 0x04}, // ^
79      {0x00, 0x40, 0x40, 0x40, 0x40, 0x40}, // _
80      {0x00, 0x00, 0x01, 0x02, 0x04, 0x00}, // '
81      {0x00, 0x20, 0x54, 0x54, 0x54, 0x78}, // a
82      {0x00, 0x7F, 0x48, 0x44, 0x44, 0x38}, // b
83      {0x00, 0x38, 0x44, 0x44, 0x44, 0x20}, // c
84      {0x00, 0x38, 0x44, 0x44, 0x48, 0x7F}, // d
85      {0x00, 0x38, 0x54, 0x54, 0x54, 0x18}, // e
86      {0x00, 0x08, 0x7E, 0x09, 0x01, 0x02}, // f
87      {0x00, 0x18, 0xA4, 0xA4, 0xA4, 0x7C}, // g
88      {0x00, 0x7F, 0x08, 0x04, 0x04, 0x78}, // h
89      {0x00, 0x00, 0x44, 0x7D, 0x40, 0x00}, // i
90      {0x00, 0x40, 0x80, 0x84, 0x7D, 0x00}, // j
91      {0x00, 0x7F, 0x10, 0x28, 0x44, 0x00}, // k
92      {0x00, 0x00, 0x41, 0x7F, 0x40, 0x00}, // l
93      {0x00, 0x7C, 0x04, 0x18, 0x04, 0x78}, // m
94      {0x00, 0x7C, 0x08, 0x04, 0x04, 0x78}, // n
95      {0x00, 0x38, 0x44, 0x44, 0x44, 0x38}, // o
96      {0x00, 0xFC, 0x24, 0x24, 0x24, 0x18}, // p
97      {0x00, 0x18, 0x24, 0x24, 0x18, 0xFC}, // q
98      {0x00, 0x7C, 0x08, 0x04, 0x04, 0x08}, // r
99      {0x00, 0x48, 0x54, 0x54, 0x54, 0x20}, // s
100      {0x00, 0x04, 0x3F, 0x44, 0x40, 0x20}, // t
101      {0x00, 0x3C, 0x40, 0x40, 0x20, 0x7C}, // u
102      {0x00, 0x1C, 0x20, 0x40, 0x20, 0x1C}, // v
103      {0x00, 0x3C, 0x40, 0x30, 0x40, 0x3C}, // w
104      {0x00, 0x44, 0x28, 0x10, 0x28, 0x44}, // x
105      {0x00, 0x1C, 0xA0, 0xA0, 0xA0, 0x7C}, // y
106      {0x00, 0x44, 0x64, 0x54, 0x4C, 0x44}, // z
107      {0x00, 0x00, 0x08, 0x77, 0x41, 0x00}, // {
108      {0x00, 0x00, 0x00, 0x63, 0x00, 0x00}, // |
109      {0x00, 0x00, 0x41, 0x77, 0x08, 0x00}, // }
110      {0x00, 0x08, 0x04, 0x08, 0x08, 0x04}, // ~
111      /* end of normal char-set */
112      /* put your own signs/chars here, edit special_char too */
113      /* be sure that your first special char stand here */
114      {0x00, 0x3A, 0x40, 0x40, 0x20, 0x7A}, // ü, !!!  Important:  this must be special_char[0] !!!
115      {0x00, 0x3D, 0x40, 0x40, 0x40, 0x3D}, // Ü
```

```
116     {0x00, 0x21, 0x54, 0x54, 0x54, 0x79}, // ä
117     {0x00, 0x7D, 0x12, 0x11, 0x12, 0x7D}, // Ä
118     {0x00, 0x39, 0x44, 0x44, 0x44, 0x39}, // ö
119     {0x00, 0x3D, 0x42, 0x42, 0x42, 0x3D}, // Ö
120     {0x00, 0x02, 0x05, 0x02, 0x00, 0x00}, // °
121     {0x00, 0x7E, 0x01, 0x49, 0x55, 0x73}, // ß
122     {0x00, 0x7C, 0x10, 0x10, 0x08, 0x1C}, // µ
123     {0x00, 0x30, 0x48, 0x20, 0x48, 0x30}, //
124     {0x00, 0x5C, 0x62, 0x02, 0x62, 0x5C} //
125 };
126
127 const char special_char[][2] PROGMEM = {
128     // define position of special char in font
129     // {special char, position in font}
130     // be sure that last element of this
131     // array are {0xff, 0xff} and first element
132     // are {first special char, first element after normal char-set in font}
133     {'ü', 95},  // special_char[0]
134     {'Ü', 96},
135     {'ä', 97},
136     {'Ä', 98},
137     {'ö', 99},
138     {'Ö', 100},
139     {'°', 101},
140     {'ß', 102},
141     {'µ', 103},
142     {'', 104},
143     {'', 105},
144     {0xff, 0xff} // end of table special_char
145 };
146
147 #endif
```

6.17 lib/oled/oled.c File Reference

```
#include "oled.h"
#include "font.h"
#include <string.h>
#include <stdlib.h>
```

Include dependency graph for oled.c:



Functions

- void oled_command (uint8_t cmd[ ], uint8_t size)

- void oled_data (uint8_t data[ ], uint16_t size)
- void oled_init (uint8_t dispAttr)
- void oled_gotoxy (uint8_t x, uint8_t y)
- void oled_goto_xpix_y (uint8_t x, uint8_t y)
- void oled_clrscr (void)
- void oled_home (void)
- void oled_invert (uint8_t invert)
- void oled_sleep (uint8_t sleep)
- void oled_set_contrast (uint8_t contrast)
- void oled_putc (char c)
- void oled_charMode (uint8_t mode)
- void oled_flip (uint8_t flipping)
- void oled_puts (const char *s)
- void oled_puts_p (const char *progmem_s)
- uint8_t oled_drawPixel (uint8_t x, uint8_t y, uint8_t color)
- uint8_t oled_drawLine (uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, uint8_t color)
- uint8_t oled_drawRect (uint8_t px1, uint8_t py1, uint8_t px2, uint8_t py2, uint8_t color)
- uint8_t oled_fillRect (uint8_t px1, uint8_t py1, uint8_t px2, uint8_t py2, uint8_t color)
- uint8_t oled_drawCircle (uint8_t center_x, uint8_t center_y, uint8_t radius, uint8_t color)
- uint8_t oled_fillCircle (uint8_t center_x, uint8_t center_y, uint8_t radius, uint8_t color)
- uint8_t oled_drawBitmap (uint8_t x, uint8_t y, const uint8_t *picture, uint8_t width, uint8_t height, uint8_t color)
- void oled_display ()
- void oled_clear_buffer ()
- uint8_t oled_check_buffer (uint8_t x, uint8_t y)
- void oled_display_block (uint8_t x, uint8_t line, uint8_t width)

Variables

- struct {
  uint8_t x
  uint8_t y
  } cursorPosition

- static uint8_t charMode = NORMALSIZE
- static uint8_t displayBuffer [DISPLAY_HEIGHT/8][DISPLAY_WIDTH]
- const uint8_t init_sequence[ ] PROGMEM

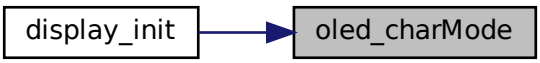
6.17.1 Function Documentation

6.17.1.1 oled_charMode()

```
void oled_charMode (
            uint8_t mode )
378                                 {
379     charMode = mode;
380 }
```

Referenced by display_init().

Here is the caller graph for this function:



6.17.1.2 oled_check_buffer()

```
uint8_t oled_check_buffer (
            uint8_t x,
            uint8_t y )
550                                                 {
551     if( x > DISPLAY_WIDTH-1 || y > (DISPLAY_HEIGHT-1)) return 0; // out of Display
552     return displayBuffer[(y / (DISPLAY_HEIGHT/8))][x] & (1 « (y % (DISPLAY_HEIGHT/8)));
553 }
```

6.17.1.3 oled_clear_buffer()

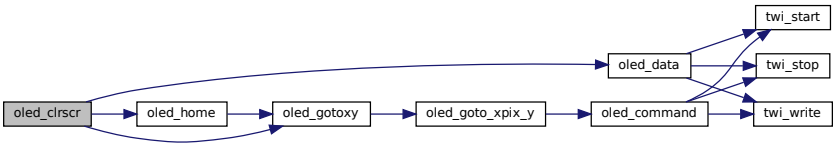
```
void oled_clear_buffer (
            void  )
545                             {
546     for (uint8_t i = 0; i < DISPLAY_HEIGHT/8; i++){
547         memset(displayBuffer[i], 0x00, sizeof(displayBuffer[i]));
548     }
549 }
```

6.17.1.4 oled_clrscr()

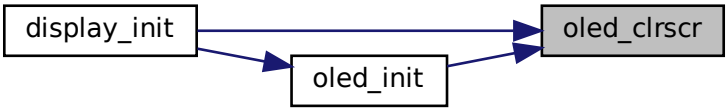
```
void oled_clrscr (
              void  )
185                       {
186 #ifdef GRAPHICMODE
187     for (uint8_t i = 0; i < DISPLAY_HEIGHT/8; i++){
188         memset(displayBuffer[i], 0x00, sizeof(displayBuffer[i]));
189         oled_gotoxy(0,i);
190         oled_data(displayBuffer[i], sizeof(displayBuffer[i]));
191     }
192 #elif defined TEXTMODE
193     uint8_t displayBuffer[DISPLAY_WIDTH];
194     memset(displayBuffer, 0x00, sizeof(displayBuffer));
195     for (uint8_t i = 0; i < DISPLAY_HEIGHT/8; i++){
196         oled_gotoxy(0,i);
197         oled_data(displayBuffer, sizeof(displayBuffer));
198     }
199 #endif
200     oled_home();
201 }
```

Referenced by display_init(), and oled_init().

Here is the call graph for this function:



Here is the caller graph for this function:



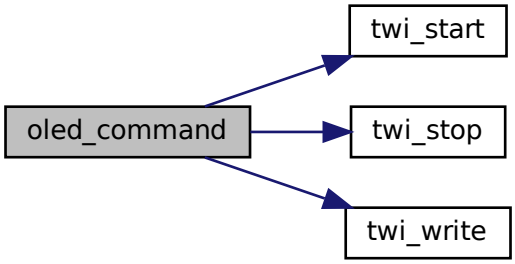
6.17.1.5 oled_command()

```
void oled_command (
              uint8_t cmd[],
              uint8_t size )
101                                           {
102 #if defined I2C
103     twi_start();
104     twi_write((OLED_I2C_ADR«1) | TWI_WRITE);
105     // i2c_start((OLED_I2C_ADR « 1) | 0);
```

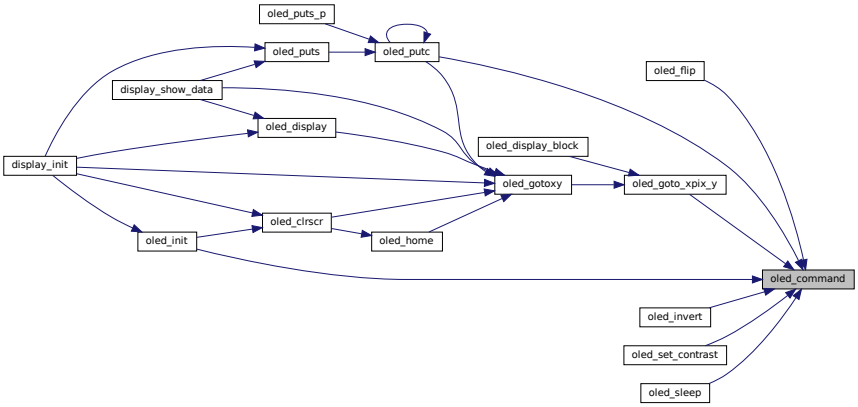
```
106     twi_write(0x00);
107     // i2c_byte(0x00);     // 0x00 for command, 0x40 for data
108     for (uint8_t i=0; i<size; i++) {
109         twi_write(cmd[i]);
110         // i2c_byte(cmd[i]);
111     }
112     twi_stop();
113 #elif defined SPI
114     OLED_PORT &= ~(1 « CS_PIN);
115     OLED_PORT &= ~(1 « DC_PIN);
116     for (uint8_t i=0; i<size; i++) {
117         SPDR = cmd[i];
118         while(!(SPSR & (1«SPIF)));
119     }
120     OLED_PORT |= (1 « CS_PIN);
121 #endif
122 }
```

Referenced by oled_flip(), oled_goto_xpix_y(), oled_init(), oled_invert(), oled_putc(), oled_set_contrast(), and oled_sleep().

Here is the call graph for this function:



Here is the caller graph for this function:

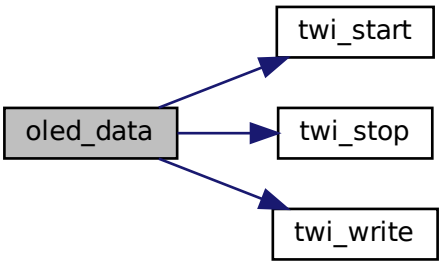


6.17.1.6 oled_data()

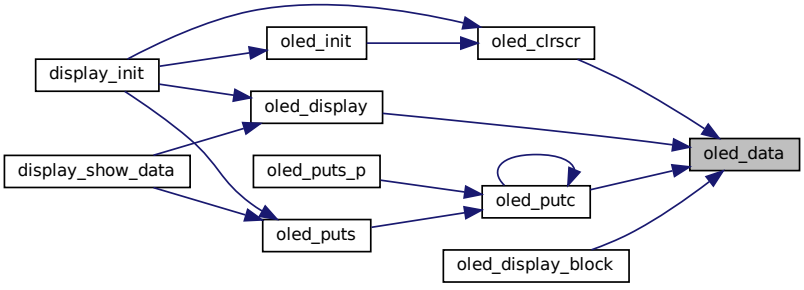
```
void oled_data (
             uint8_t data[],
             uint16_t size )
123                                          {
124 #if defined I2C
125     twi_start();
126     twi_write((OLED_I2C_ADR«1) | TWI_WRITE);
127     // i2c_start((OLED_I2C_ADR « 1) | 0);
128     twi_write(0x40);
129     // i2c_byte(0x40);    // 0x00 for command, 0x40 for data
130     for (uint16_t i = 0; i<size; i++) {
131         twi_write(data[i]);
132         // i2c_byte(data[i]);
133     }
134     twi_stop();
135     // i2c_stop();
136 #elif defined SPI
137     OLED_PORT &= ~(1 « CS_PIN);
138     OLED_PORT |= (1 « DC_PIN);
139     for (uint16_t i = 0; i<size; i++) {
140         SPDR = data[i];
141         while(!(SPSR & (1«SPIF)));
142     }
143     OLED_PORT |= (1 « CS_PIN);
144 #endif
145 }
```

Referenced by oled_clrscr(), oled_display(), oled_display_block(), and oled_putc().

Here is the call graph for this function:



Here is the caller graph for this function:

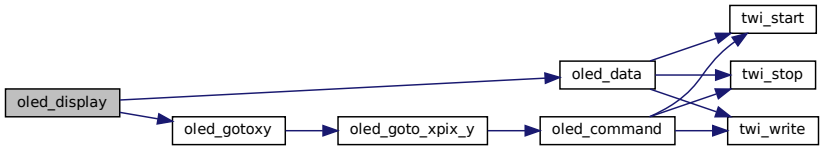


6.17.1.7 oled_display()

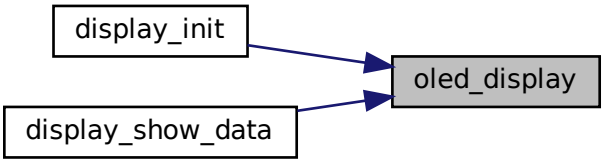
```
void oled_display (
              void  )
534                   {
535 #if defined (SSD1306) || defined (SSD1309)
536     oled_gotoxy(0,0);
537     oled_data(&displayBuffer[0][0], DISPLAY_WIDTH*DISPLAY_HEIGHT/8);
538 #elif defined SH1106
539     for (uint8_t i = 0; i < DISPLAY_HEIGHT/8; i++){
540         oled_gotoxy(0,i);
541         oled_data(displayBuffer[i], sizeof(displayBuffer[i]));
542     }
543 #endif
544 }
```

Referenced by display_init(), and display_show_data().

Here is the call graph for this function:



Here is the caller graph for this function:

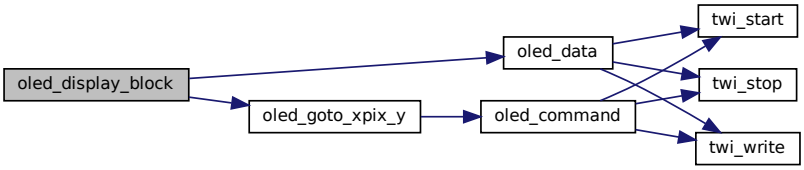


6.17.1.8 oled_display_block()

```
void oled_display_block (
              uint8_t x,
              uint8_t line,
              uint8_t width )
554                                                     {
555     if (line > (DISPLAY_HEIGHT/8-1) || x > DISPLAY_WIDTH - 1){return;}
```

```
556     if (x + width > DISPLAY_WIDTH) { // no -1 here, x alone is width 1
557         width = DISPLAY_WIDTH - x;
558     }
559     oled_goto_xpix_y(x,line);
560     oled_data(&displayBuffer[line][x], width);
561 }
```

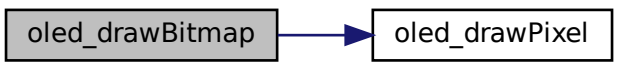
Here is the call graph for this function:



6.17.1.9 oled_drawBitmap()

```
uint8_t oled_drawBitmap (
              uint8_t x,
              uint8_t y,
              const uint8_t * picture,
              uint8_t width,
              uint8_t height,
              uint8_t color )
521
      {
522     uint8_t result,i,j, byteWidth = (width+7)/8;
523     for (j = 0; j < height; j++) {
524         for(i=0; i < width;i++){
525             if(pgm_read_byte(picture + j * byteWidth + i / 8) & (128 » (i & 7))){
526                 result = oled_drawPixel(x+i, y+j, color);
527             } else {
528                 result = oled_drawPixel(x+i, y+j, !color);
529             }
530         }
531     }
532     return result;
533 }
```

Here is the call graph for this function:



6.17.1.10 oled_drawCircle()

```
uint8_t oled_drawCircle (
              uint8_t center_x,
              uint8_t center_y,
              uint8_t radius,
              uint8_t color )
```

```
479                                                                                                 {
480     uint8_t result;
481
482     int16_t f = 1 - radius;
483     int16_t ddF_x = 1;
484     int16_t ddF_y = -2 * radius;
485     int16_t x = 0;
486     int16_t y = radius;
487
488     result = oled_drawPixel(center_x  , center_y+radius, color);
489     result = oled_drawPixel(center_x  , center_y-radius, color);
490     result = oled_drawPixel(center_x+radius, center_y  , color);
491     result = oled_drawPixel(center_x-radius, center_y  , color);
492
493     while (x<y) {
494         if (f >= 0) {
495             y--;
496             ddF_y += 2;
497             f += ddF_y;
498         }
499         x++;
500         ddF_x += 2;
501         f += ddF_x;
502
503         result = oled_drawPixel(center_x + x, center_y + y, color);
504         result = oled_drawPixel(center_x - x, center_y + y, color);
505         result = oled_drawPixel(center_x + x, center_y - y, color);
506         result = oled_drawPixel(center_x - x, center_y - y, color);
507         result = oled_drawPixel(center_x + y, center_y + x, color);
508         result = oled_drawPixel(center_x - y, center_y + x, color);
509         result = oled_drawPixel(center_x + y, center_y - x, color);
510         result = oled_drawPixel(center_x - y, center_y - x, color);
511     }
512     return result;
513 }
```

Referenced by oled_fillCircle().

Here is the call graph for this function:

Here is the caller graph for this function:

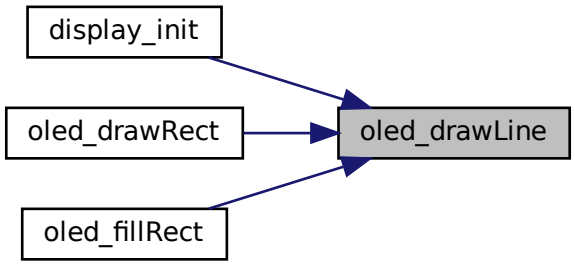
6.17.1.11 oled_drawLine()

```
uint8_t oled_drawLine (
              uint8_t x1,
              uint8_t y1,
              uint8_t x2,
              uint8_t y2,
              uint8_t color )
435                                                                                 {
436     uint8_t result;
437
438     int dx =  abs(x2-x1), sx = x1<x2 ?  1 :  -1;
439     int dy = -abs(y2-y1), sy = y1<y2 ?  1 :  -1;
440     int err = dx+dy, e2; /* error value e_xy */
441
442     while(1){
443         result = oled_drawPixel(x1, y1, color);
444         if (x1==x2 && y1==y2) break;
445         e2 = 2*err;
446         if (e2 > dy) { err += dy; x1 += sx; } /* e_xy+e_x > 0 */
447         if (e2 < dx) { err += dx; y1 += sy; } /* e_xy+e_y < 0 */
448     }
449
450     return result;
451 }
```

Referenced by display_init(), oled_drawRect(), and oled_fillRect().

Here is the call graph for this function:

Here is the caller graph for this function:

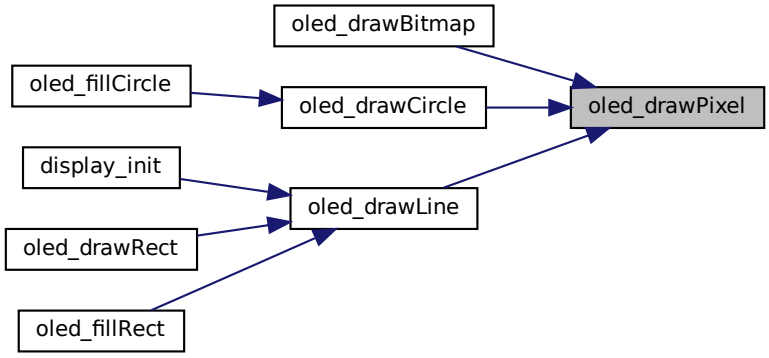


6.17.1.12 oled_drawPixel()

```
uint8_t oled_drawPixel (
              uint8_t x,
              uint8_t y,
              uint8_t color )
424                                                                 {
425     if( x > DISPLAY_WIDTH-1 || y > (DISPLAY_HEIGHT-1)) return 1; // out of Display
426
427     if( color == WHITE){
428         displayBuffer[(y / 8)][x] |= (1 « (y % 8));
429     } else {
430         displayBuffer[(y / 8)][x] &= ~(1 « (y % 8));
431     }
432
433     return 0;
434 }
```

Referenced by oled_drawBitmap(), oled_drawCircle(), and oled_drawLine().

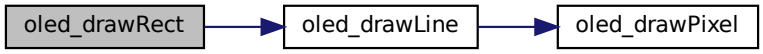
Here is the caller graph for this function:



6.17.1.13 oled_drawRect()

```
uint8_t oled_drawRect (
              uint8_t px1,
              uint8_t py1,
              uint8_t px2,
              uint8_t py2,
              uint8_t color )
452                                                                                 {
453     uint8_t result;
454
455     result = oled_drawLine(px1, py1, px2, py1, color);
456     result = oled_drawLine(px2, py1, px2, py2, color);
457     result = oled_drawLine(px2, py2, px1, py2, color);
458     result = oled_drawLine(px1, py2, px1, py1, color);
459
460     return result;
461 }
```

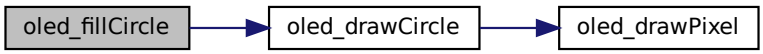
Here is the call graph for this function:



6.17.1.14 oled_fillCircle()

```
uint8_t oled_fillCircle (
              uint8_t center_x,
              uint8_t center_y,
              uint8_t radius,
              uint8_t color )
514                                                                                 {
515     uint8_t result;
516     for(uint8_t i=0; i<= radius;i++){
517         result = oled_drawCircle(center_x, center_y, i, color);
518     }
519     return result;
520 }
```

Here is the call graph for this function:

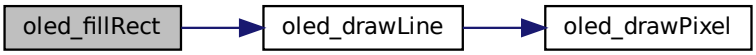


6.17.1.15 oled_fillRect()

```
uint8_t oled_fillRect (
              uint8_t px1,
              uint8_t py1,
              uint8_t px2,
              uint8_t py2,
              uint8_t color )
462                                                                                 {
463     uint8_t result;
464
465     if( px1 > px2){
466         uint8_t temp = px1;
467         px1 = px2;
468         px2 = temp;
469         temp = py1;
470         py1 = py2;
471         py2 = temp;
```

```
472     }
473     for (uint8_t i=0; i<=(py2-py1); i++){
474         result = oled_drawLine(px1, py1+i, px2, py1+i, color);
475     }
476 
477     return result;
478 }
```

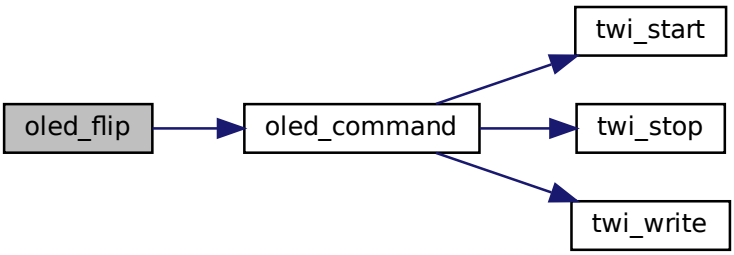
Here is the call graph for this function:



6.17.1.16 oled_flip()

```
void oled_flip (
              uint8_t flipping )
381                                 {
382     uint8_t command[2] = {0xC8, 0xA1};
383     switch(flipping){
384         case 0:
385             // normal mode default at init (needs to be reload data to display)
386             command[0] = 0xC8;
387             command[1] = 0xA1;
388             oled_command(command, sizeof(command));
389             break;
390         case 1:
391             // flip horizontal && vertical (needs to be reload data to display)
392             command[0] = 0xC0;
393             command[1] = 0xA0;
394             oled_command(command, sizeof(command));
395             break;
396         case 2:
397             // flip vertical (immediate without reload data to display)
398             command[0] = 0xC0;
399             oled_command(command, sizeof(command));
400             break;
401         case 3:
402             // flip horizontal (needs to be reload data to display)
403             command[1] = 0xA0;
404             oled_command(command, sizeof(command));
405         default:
406             // do nothing
407             break;
408     }
409 }
```

Here is the call graph for this function:

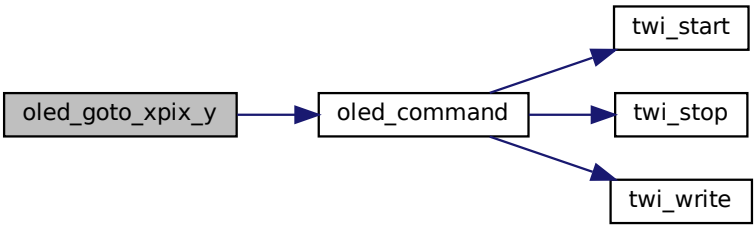


6.17.1.17 oled_goto_xpix_y()

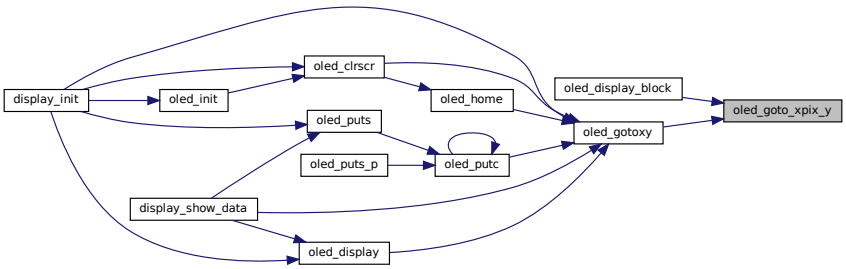
```
void oled_goto_xpix_y (
             uint8_t x,
             uint8_t y )
174                                          {
175     if( x > (DISPLAY_WIDTH) || y > (DISPLAY_HEIGHT/8-1)) return;// out of display
176     cursorPosition.x=x;
177     cursorPosition.y=y;
178 #if defined (SSD1306) || defined (SSD1309)
179     uint8_t commandSequence[] = {0xb0+y, 0x21, x, 0x7f};
180 #elif defined SH1106
181     uint8_t commandSequence[] = {0xb0+y, 0x21, 0x00+((2+x) & (0x0f)), 0x10+( ((2+x) & (0xf0)) » 4 ),
      0x7f};
182 #endif
183     oled_command(commandSequence, sizeof(commandSequence));
184 }
```

Referenced by oled_display_block(), and oled_gotoxy().

Here is the call graph for this function:



Here is the caller graph for this function:

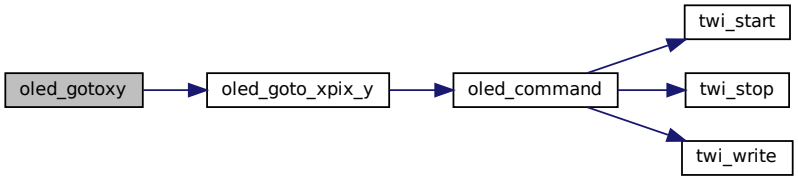


6.17.1.18 oled_gotoxy()

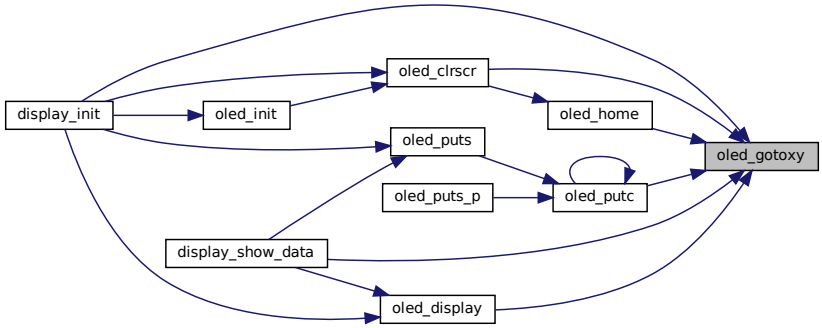
```
void oled_gotoxy (
             uint8_t x,
             uint8_t y )
170                                     {
171     x = x * sizeof(FONT[0]);
172     oled_goto_xpix_y(x,y);
173 }
```

Referenced by display_init(), display_show_data(), oled_clrscr(), oled_display(), oled_home(), and oled_putc().

Here is the call graph for this function:



Here is the caller graph for this function:



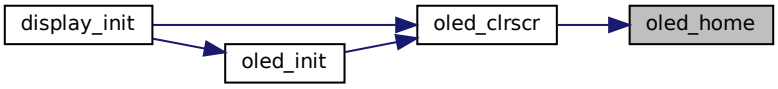
6.17.1.19 oled_home()

```
void oled_home (
            void  )
202                     {
203     oled_gotoxy(0, 0);
204 }
```

Referenced by oled_clrscr().

Here is the call graph for this function:

Here is the caller graph for this function:



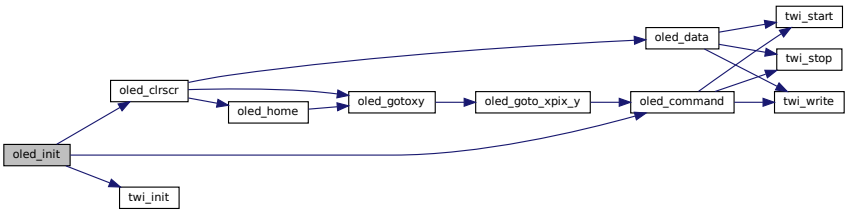
6.17.1.20 oled_init()

```
void oled_init (
            uint8_t dispAttr )
148                                 {
149 #if defined I2C
150     // i2c_init();
151     twi_init();
152 #elif defined SPI
153     DDRB |= (1 « PB2)|(1 « PB3)|(1 « PB5);
154     SPCR = (1 « SPE)|(1«MSTR)|(1«SPR0);
155     OLED_DDR |= (1 « CS_PIN)|(1 « DC_PIN)|(1 « RES_PIN);
156     OLED_PORT |= (1 « CS_PIN)|(1 « DC_PIN)|(1 « RES_PIN);
157     OLED_PORT &= ~(1 « RES_PIN);
158     _delay_ms(10);
159     OLED_PORT |= (1 « RES_PIN);
160 #endif
161
162     uint8_t commandSequence[sizeof(init_sequence)+1];
163     for (uint8_t i = 0; i < sizeof (init_sequence); i++) {
164         commandSequence[i] = (pgm_read_byte(&init_sequence[i]));
165     }
```

```
166     commandSequence[sizeof(init_sequence)]=(dispAttr);
167     oled_command(commandSequence, sizeof(commandSequence));
168     oled_clrscr();
169 }
```

Referenced by display_init().

Here is the call graph for this function:

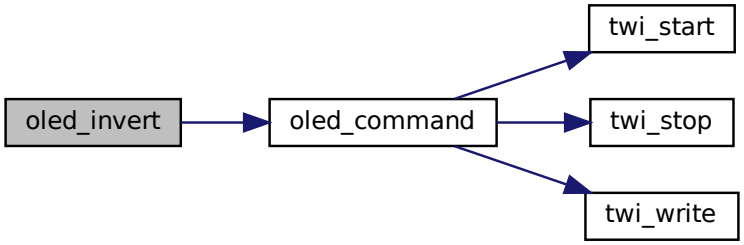


Here is the caller graph for this function:

6.17.1.21 oled_invert()

```
void oled_invert (
             uint8_t invert )
205                                 {
206     uint8_t commandSequence[1];
207     if (invert != YES) {
208         commandSequence[0] = 0xA6;
209     } else {
210         commandSequence[0] = 0xA7;
211     }
212     oled_command(commandSequence, 1);
213 }
```

Here is the call graph for this function:



6.17.1.22 oled_putc()

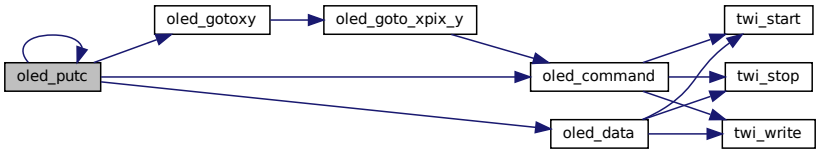
```
void oled_putc (
             char c )
227                     {
228     switch (c) {
229         case '\b':
230             // backspace
231             oled_gotoxy(cursorPosition.x-charMode, cursorPosition.y);
232             oled_putc(' ');
233             oled_gotoxy(cursorPosition.x-charMode, cursorPosition.y);
234             break;
235         case '\t':
236             // tab
237             if( (cursorPosition.x+charMode*4) < (DISPLAY_WIDTH/ sizeof(FONT[0])-charMode*4) ){
238                 oled_gotoxy(cursorPosition.x+charMode*4, cursorPosition.y);
239             }else{
240                 oled_gotoxy(DISPLAY_WIDTH/ sizeof(FONT[0]), cursorPosition.y);
241             }
242             break;
243         case '\n':
244             // linefeed
245             if(cursorPosition.y < (DISPLAY_HEIGHT/8-1)){
246                 oled_gotoxy(cursorPosition.x, cursorPosition.y+charMode);
247             }
248             break;
249         case '\r':
250             // carrige return
251             oled_gotoxy(0, cursorPosition.y);
252             break;
253         default:
254             // char doesn't fit in line
255             if( (cursorPosition.x >= DISPLAY_WIDTH-sizeof(FONT[0])) || (c < ' ') ) break;
256             // mapping char
257             c -= ' ';
258             if (c >= pgm_read_byte(&special_char[0][1]) ) {
259                 char temp = c;
260                 c = 0xff;
261                 for (uint8_t i=0; pgm_read_byte(&special_char[i][1]) != 0xff; i++) {
262                     if ( pgm_read_byte(&special_char[i][0])-' ' == temp ) {
263                         c = pgm_read_byte(&special_char[i][1]);
264                         break;
265                     }
266                 }
267                 if ( c == 0xff ) break;
268             }
269             // print char at display
270 #ifdef GRAPHICMODE
271             if (charMode == DOUBLESIZE) {
272                 uint16_t doubleChar[sizeof(FONT[0])];
273                 uint8_t dChar;
274                 if ((cursorPosition.x+2*sizeof(FONT[0]))>DISPLAY_WIDTH) break;
```

```
275
276                 for (uint8_t i=0; i < sizeof(FONT[0]); i++) {
277                     doubleChar[i] = 0;
278                     dChar = pgm_read_byte(&(FONT[(uint8_t)c][i]));
279                     for (uint8_t j=0; j<8; j++) {
280                         if ((dChar & (1 « j))) {
281                             doubleChar[i] |= (1 « (j*2));
282                             doubleChar[i] |= (1 « ((j*2)+1));
283                         }
284                     }
285                 }
286                 for (uint8_t i = 0; i < sizeof(FONT[0]); i++)
287                 {
288                     // load bit-pattern from flash
289                     displayBuffer[cursorPosition.y+1][cursorPosition.x+(2*i)] = doubleChar[i] » 8;
290                     displayBuffer[cursorPosition.y+1][cursorPosition.x+(2*i)+1] = doubleChar[i] » 8;
291                     displayBuffer[cursorPosition.y][cursorPosition.x+(2*i)] = doubleChar[i] & 0xff;
292                     displayBuffer[cursorPosition.y][cursorPosition.x+(2*i)+1] = doubleChar[i] & 0xff;
293                 }
294                 cursorPosition.x += sizeof(FONT[0])*2;
295             } else {
296                 if ((cursorPosition.x+sizeof(FONT[0]))>DISPLAY_WIDTH) break;
297
298                 for (uint8_t i = 0; i < sizeof(FONT[0]); i++)
299                 {
300                     // load bit-pattern from flash
301                     displayBuffer[cursorPosition.y][cursorPosition.x+i]
    =pgm_read_byte(&(FONT[(uint8_t)c][i]));
302                 }
303                 cursorPosition.x += sizeof(FONT[0]);
304             }
305 #elif defined TEXTMODE
306             if (charMode == DOUBLESIZE) {
307                 uint16_t doubleChar[sizeof(FONT[0])];
308                 uint8_t dChar;
309                 if ((cursorPosition.x+2*sizeof(FONT[0]))>DISPLAY_WIDTH) break;
310
311                 for (uint8_t i=0; i < sizeof(FONT[0]); i++) {
312                     doubleChar[i] = 0;
313                     dChar = pgm_read_byte(&(FONT[(uint8_t)c][i]));
314                     for (uint8_t j=0; j<8; j++) {
315                         if ((dChar & (1 « j))) {
316                             doubleChar[i] |= (1 « (j*2));
317                             doubleChar[i] |= (1 « ((j*2)+1));
318                         }
319                     }
320                 }
321                 uint8_t data[sizeof(FONT[0])*2];
322                 for (uint8_t i = 0; i < sizeof(FONT[0]); i++)
323                 {
324                     // print font to ram, print 6 columns
325                     data[i«1]=(doubleChar[i] & 0xff);
326                     data[(i«1)+1]=(doubleChar[i] & 0xff);
327                 }
328                 oled_data(data, sizeof(FONT[0])*2);
329
330 #if defined (SSD1306) || defined (SSD1309)
331                 uint8_t commandSequence[] = {0xb0+cursorPosition.y+1,
332                     0x21,
333                     cursorPosition.x,
334                     0x7f};
335 #elif defined SH1106
336                 uint8_t commandSequence[] = {0xb0+cursorPosition.y+1,
337                     0x21,
338                     0x00+((2+cursorPosition.x) & (0x0f)),
339                     0x10+( ((2+cursorPosition.x) & (0xf0)) » 4 ),
340                     0x7f};
341 #endif
342                 oled_command(commandSequence, sizeof(commandSequence));
343
344                 for (uint8_t i = 0; i < sizeof(FONT[0]); i++)
345                 {
346                     // print font to ram, print 6 columns
347                     data[i«1]=(doubleChar[i] » 8);
348                     data[(i«1)+1]=(doubleChar[i] » 8);
349                 }
350                 oled_data(data, sizeof(FONT[0])*2);
351
352                 commandSequence[0] = 0xb0+cursorPosition.y;
353 #if defined (SSD1306) || defined (SSD1309)
354                 commandSequence[2] = cursorPosition.x+(2*sizeof(FONT[0]));
355 #elif defined SH1106
356                 commandSequence[2] = 0x00+((2+cursorPosition.x+(2*sizeof(FONT[0]))) & (0x0f));
357                 commandSequence[3] = 0x10+( ((2+cursorPosition.x+(2*sizeof(FONT[0]))) & (0xf0)) » 4 );
358 #endif
359                 oled_command(commandSequence, sizeof(commandSequence));
360                 cursorPosition.x += sizeof(FONT[0])*2;
```

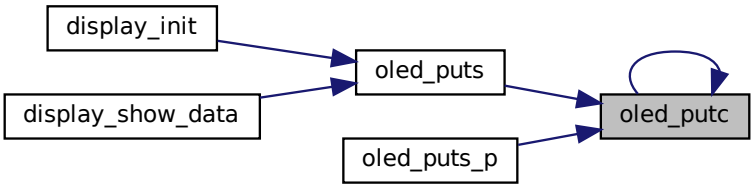
```
361             } else {
362                 uint8_t data[sizeof(FONT[0])];
363                 if ((cursorPosition.x+sizeof(FONT[0]))>DISPLAY_WIDTH) break;
364 
365                 for (uint8_t i = 0; i < sizeof(FONT[0]); i++)
366                 {
367                     // print font to ram, print 6 columns
368                     data[i]=(pgm_read_byte(&(FONT[(uint8_t)c][i])));
369                 }
370                 oled_data(data, sizeof(FONT[0]));
371                 cursorPosition.x += sizeof(FONT[0]);
372             }
373 #endif
374             break;
375     }
376 
377 }
```

Referenced by oled_putc(), oled_puts(), and oled_puts_p().

Here is the call graph for this function:



Here is the caller graph for this function:

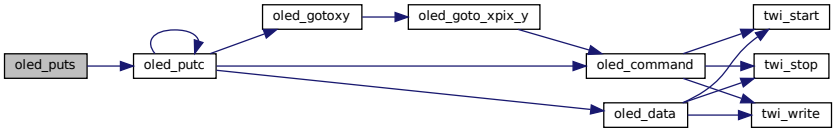


6.17.1.23 oled_puts()

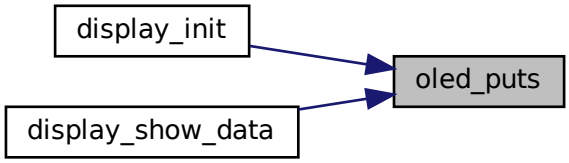
```
void oled_puts (
             const char * s )
410                              {
411     while (*s) {
412         oled_putc(*s++);
413     }
414 }
```

Referenced by display_init(), and display_show_data().

Here is the call graph for this function:



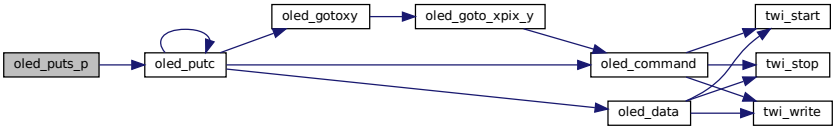
Here is the caller graph for this function:



6.17.1.24 oled_puts_p()

```
void oled_puts_p (
              const char * progmem_s )
415                                     {
416     register uint8_t c;
417     while ((c = pgm_read_byte(progmem_s++))) {
418         oled_putc(c);
419     }
420 }
```

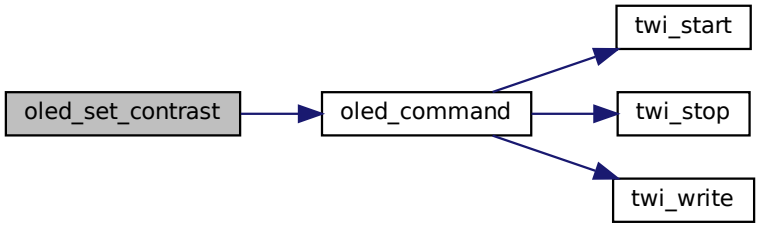
Here is the call graph for this function:



6.17.1.25 oled_set_contrast()

```
void oled_set_contrast (
             uint8_t contrast )
223                                     {
224     uint8_t commandSequence[2] = {0x81, contrast};
225     oled_command(commandSequence, sizeof(commandSequence));
226 }
```

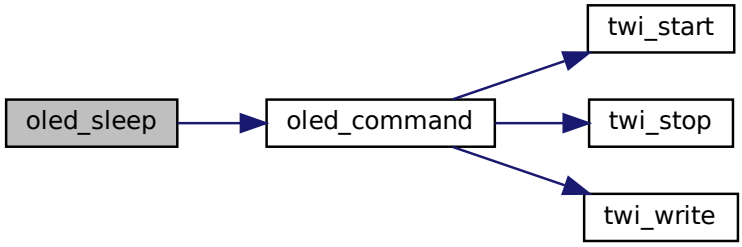
Here is the call graph for this function:



6.17.1.26 oled_sleep()

```
void oled_sleep (
             uint8_t sleep )
214                                 {
215     uint8_t commandSequence[1];
216     if (sleep != YES) {
217         commandSequence[0] = 0xAF;
218     } else {
219         commandSequence[0] = 0xAE;
220     }
221     oled_command(commandSequence, 1);
222 }
```

Here is the call graph for this function:



6.17.2 Variable Documentation

6.17.2.1 charMode

```
uint8_t charMode = NORMALSIZE  [static]
```

Referenced by oled_charMode(), and oled_putc().

6.17.2.2

```
struct { ...  } cursorPosition  [static]
```

Referenced by oled_goto_xpix_y(), and oled_putc().

6.17.2.3 displayBuffer

```
uint8_t displayBuffer[DISPLAY_HEIGHT/8][DISPLAY_WIDTH]  [static]
```

Referenced by oled_check_buffer(), oled_clear_buffer(), oled_clrscr(), oled_display(), oled_display_block(), oled_drawPixel(), and oled_putc().

6.17.2.4 PROGMEM

```
const uint8_t init_sequence [ ] PROGMEM
```

Initial value:

```
= {
    OLED_DISP_OFF,
    0x20, 0b00,


    0xB0,
    0xC8,
    0x00,
    0x10,
    0x40,
    0x81, 0x3F,
    0xA1,
    0xA6,
    0xA8, DISPLAY_HEIGHT-1,
    0xA4,

    0xD3, 0x00,
    0xD5,
    0xF0,
    0xD9, 0x22,

    0xDA, 0x12,
    0xDB,
    0x20,
    0x8D, 0x14,
}
```

6.17.2.5 x

```
uint8_t x
```

Referenced by oled_check_buffer(), oled_display_block(), oled_drawBitmap(), oled_drawCircle(), oled_drawPixel(), oled_goto_xpix_y(), and oled_gotoxy().

6.17.2.6 y

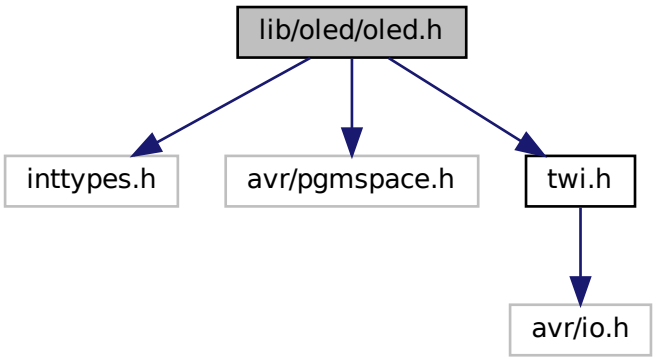
```
uint8_t y
```

Referenced by oled_check_buffer(), oled_drawBitmap(), oled_drawCircle(), oled_drawPixel(), oled_goto_xpix_y(), and oled_gotoxy().

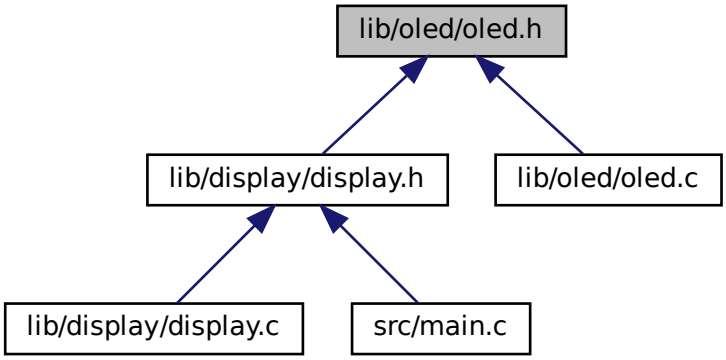
6.18 lib/oled/oled.h File Reference

```
#include <inttypes.h>
#include <avr/pgmspace.h>
#include "twi.h"
```

Include dependency graph for oled.h:



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



Macros

- #define I2C
- #define SH1106
- #define GRAPHICMODE
- #define FONT ssd1306oled_font
- #define OLED_I2C_ADR (0x3c)
- #define YES 1
- #define NORMALSIZE 1
- #define DOUBLESIZE 2
- #define OLED_DISP_OFF 0xAE
- #define OLED_DISP_ON 0xAF
- #define WHITE 0x01
- #define BLACK 0x00
- #define DISPLAY_WIDTH 128
- #define DISPLAY_HEIGHT 64

Functions

- void oled_command (uint8_t cmd[ ], uint8_t size)
- void oled_data (uint8_t data[ ], uint16_t size)
- void oled_init (uint8_t dispAttr)
- void oled_home (void)
- void oled_invert (uint8_t invert)
- void oled_sleep (uint8_t sleep)
- void oled_set_contrast (uint8_t contrast)
- void oled_puts (const char ∗s)
- void oled_puts_p (const char ∗progmem_s)
- void oled_clrscr (void)
- void oled_gotoxy (uint8_t x, uint8_t y)
- void oled_goto_xpix_y (uint8_t x, uint8_t y)
- void oled_putc (char c)

- void oled_charMode (uint8_t mode)
- void oled_flip (uint8_t flipping)
- uint8_t oled_drawPixel (uint8_t x, uint8_t y, uint8_t color)
- uint8_t oled_drawLine (uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, uint8_t color)
- uint8_t oled_drawRect (uint8_t px1, uint8_t py1, uint8_t px2, uint8_t py2, uint8_t color)
- uint8_t oled_fillRect (uint8_t px1, uint8_t py1, uint8_t px2, uint8_t py2, uint8_t color)
- uint8_t oled_drawCircle (uint8_t center_x, uint8_t center_y, uint8_t radius, uint8_t color)
- uint8_t oled_fillCircle (uint8_t center_x, uint8_t center_y, uint8_t radius, uint8_t color)
- uint8_t oled_drawBitmap (uint8_t x, uint8_t y, const uint8_t picture[], uint8_t width, uint8_t height, uint8_t color)
- void oled_display (void)
- void oled_clear_buffer (void)
- uint8_t oled_check_buffer (uint8_t x, uint8_t y)
- void oled_display_block (uint8_t x, uint8_t line, uint8_t width)

6.18.1 Macro Definition Documentation

6.18.1.1 BLACK

```
#define BLACK 0x00
```

6.18.1.2 DISPLAY_HEIGHT

```
#define DISPLAY_HEIGHT 64
```

6.18.1.3 DISPLAY_WIDTH

```
#define DISPLAY_WIDTH 128
```

6.18.1.4 DOUBLESIZE

```
#define DOUBLESIZE 2
```

6.18.1.5 FONT

```
#define FONT ssd1306oled_font
```

6.18.1.6 GRAPHICMODE

```
#define GRAPHICMODE
```

6.18.1.7 I2C

```
#define I2C
```

6.18.1.8 NORMALSIZE

```
#define NORMALSIZE 1
```

6.18.1.9 OLED_DISP_OFF

```
#define OLED_DISP_OFF 0xAE
```

6.18.1.10 OLED_DISP_ON

```
#define OLED_DISP_ON 0xAF
```

6.18.1.11 OLED_I2C_ADR

```
#define OLED_I2C_ADR (0x3c)
```

6.18.1.12 SH1106

```
#define SH1106
```

6.18.1.13 WHITE

```
#define WHITE 0x01
```

6.18.1.14 YES

```
#define YES 1
```

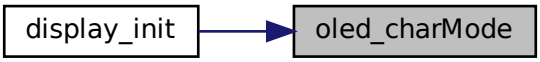
6.18.2 Function Documentation

6.18.2.1 oled_charMode()

```
void oled_charMode (
            uint8_t mode )
378                                 {
379     charMode = mode;
380 }
```

Referenced by display_init().

Here is the caller graph for this function:



6.18.2.2 oled_check_buffer()

```
uint8_t oled_check_buffer (
            uint8_t x,
            uint8_t y )
550                                                 {
551     if( x > DISPLAY_WIDTH-1 || y > (DISPLAY_HEIGHT-1)) return 0; // out of Display
552     return displayBuffer[(y / (DISPLAY_HEIGHT/8))][x] & (1 « (y % (DISPLAY_HEIGHT/8)));
553 }
```

6.18.2.3 oled_clear_buffer()

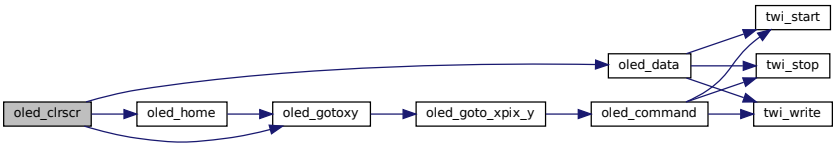
```
void oled_clear_buffer (
            void  )
545                         {
546     for (uint8_t i = 0; i < DISPLAY_HEIGHT/8; i++){
547         memset(displayBuffer[i], 0x00, sizeof(displayBuffer[i]));
548     }
549 }
```

6.18.2.4 oled_clrscr()

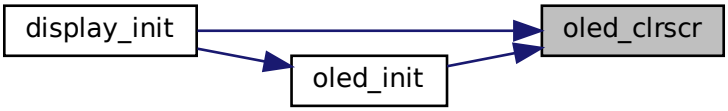
```
void oled_clrscr (
              void  )
185                       {
186 #ifdef GRAPHICMODE
187     for (uint8_t i = 0; i < DISPLAY_HEIGHT/8; i++){
188         memset(displayBuffer[i], 0x00, sizeof(displayBuffer[i]));
189         oled_gotoxy(0,i);
190         oled_data(displayBuffer[i], sizeof(displayBuffer[i]));
191     }
192 #elif defined TEXTMODE
193     uint8_t displayBuffer[DISPLAY_WIDTH];
194     memset(displayBuffer, 0x00, sizeof(displayBuffer));
195     for (uint8_t i = 0; i < DISPLAY_HEIGHT/8; i++){
196         oled_gotoxy(0,i);
197         oled_data(displayBuffer, sizeof(displayBuffer));
198     }
199 #endif
200     oled_home();
201 }
```

Referenced by display_init(), and oled_init().

Here is the call graph for this function:



Here is the caller graph for this function:



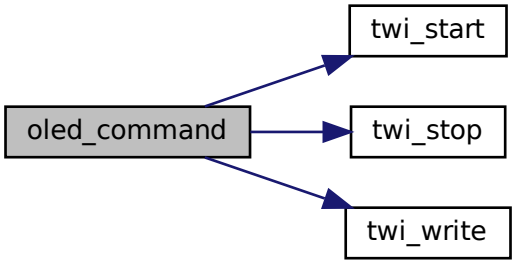
6.18.2.5 oled_command()

```
void oled_command (
              uint8_t cmd[],
              uint8_t size )
101                                           {
102 #if defined I2C
103     twi_start();
104     twi_write((OLED_I2C_ADR«1) | TWI_WRITE);
105     // i2c_start((OLED_I2C_ADR « 1) | 0);
```

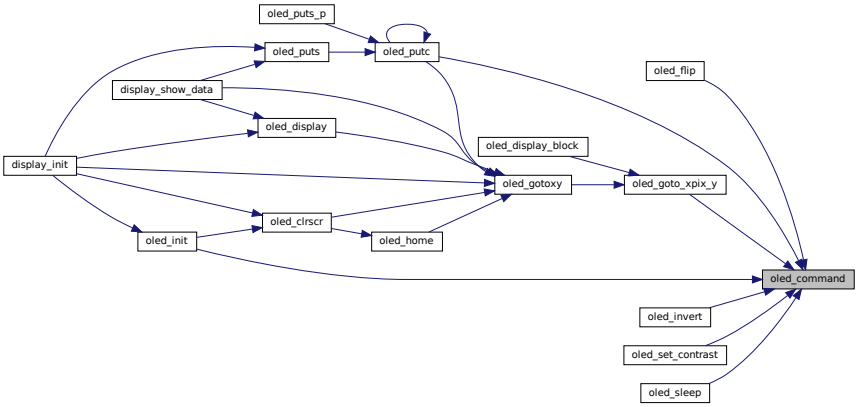
```
106     twi_write(0x00);
107     // i2c_byte(0x00);     // 0x00 for command, 0x40 for data
108     for (uint8_t i=0; i<size; i++) {
109         twi_write(cmd[i]);
110         // i2c_byte(cmd[i]);
111     }
112     twi_stop();
113 #elif defined SPI
114     OLED_PORT &= ~(1 « CS_PIN);
115     OLED_PORT &= ~(1 « DC_PIN);
116     for (uint8_t i=0; i<size; i++) {
117         SPDR = cmd[i];
118         while(!(SPSR & (1«SPIF)));
119     }
120     OLED_PORT |= (1 « CS_PIN);
121 #endif
122 }
```

Referenced by oled_flip(), oled_goto_xpix_y(), oled_init(), oled_invert(), oled_putc(), oled_set_contrast(), and oled_sleep().

Here is the call graph for this function:



Here is the caller graph for this function:

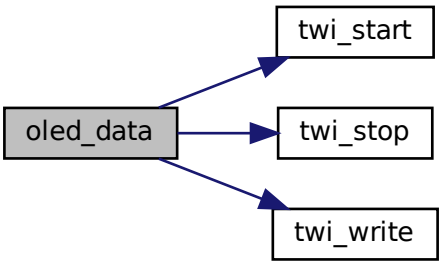


6.18.2.6 oled_data()

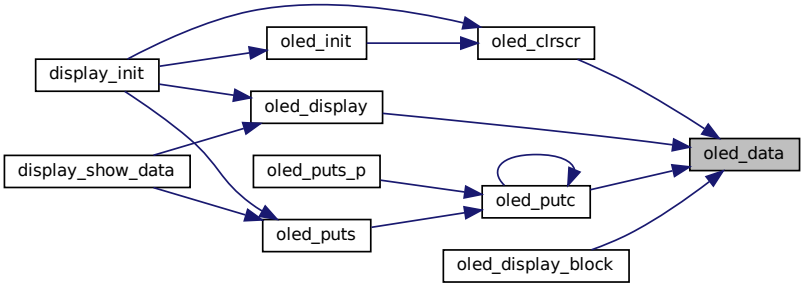
```
void oled_data (
            uint8_t data[],
            uint16_t size )
123                                             {
124 #if defined I2C
125     twi_start();
126     twi_write((OLED_I2C_ADR«1) | TWI_WRITE);
127     // i2c_start((OLED_I2C_ADR « 1) | 0);
128     twi_write(0x40);
129     // i2c_byte(0x40);    // 0x00 for command, 0x40 for data
130     for (uint16_t i = 0; i<size; i++) {
131         twi_write(data[i]);
132         // i2c_byte(data[i]);
133     }
134     twi_stop();
135     // i2c_stop();
136 #elif defined SPI
137     OLED_PORT &= ~(1 « CS_PIN);
138     OLED_PORT |= (1 « DC_PIN);
139     for (uint16_t i = 0; i<size; i++) {
140         SPDR = data[i];
141         while(!(SPSR & (1«SPIF)));
142     }
143     OLED_PORT |= (1 « CS_PIN);
144 #endif
145 }
```

Referenced by oled_clrscr(), oled_display(), oled_display_block(), and oled_putc().

Here is the call graph for this function:



Here is the caller graph for this function:

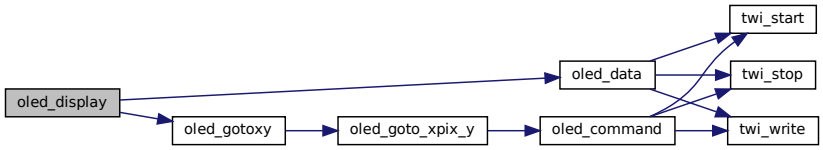


6.18.2.7 oled_display()

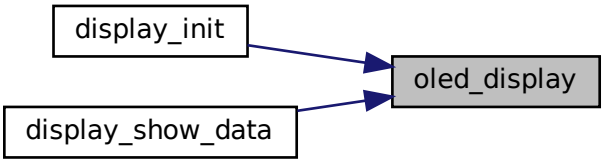
```
void oled_display (
              void  )
534                   {
535 #if defined (SSD1306) || defined (SSD1309)
536     oled_gotoxy(0,0);
537     oled_data(&displayBuffer[0][0], DISPLAY_WIDTH*DISPLAY_HEIGHT/8);
538 #elif defined SH1106
539     for (uint8_t i = 0; i < DISPLAY_HEIGHT/8; i++){
540         oled_gotoxy(0,i);
541         oled_data(displayBuffer[i], sizeof(displayBuffer[i]));
542     }
543 #endif
544 }
```

Referenced by display_init(), and display_show_data().

Here is the call graph for this function:



Here is the caller graph for this function:

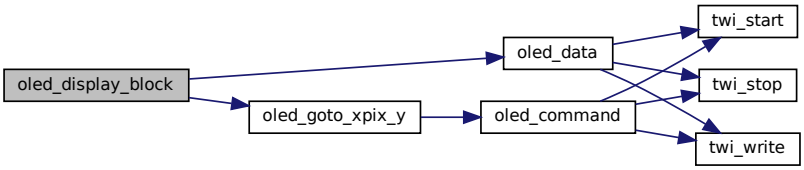


6.18.2.8 oled_display_block()

```
void oled_display_block (
              uint8_t x,
              uint8_t line,
              uint8_t width )
554                                                         {
555     if (line > (DISPLAY_HEIGHT/8-1) || x > DISPLAY_WIDTH - 1){return;}
```

```
556     if (x + width > DISPLAY_WIDTH) { // no -1 here, x alone is width 1
557         width = DISPLAY_WIDTH - x;
558     }
559     oled_goto_xpix_y(x,line);
560     oled_data(&displayBuffer[line][x], width);
561 }
```

Here is the call graph for this function:



6.18.2.9 oled_drawBitmap()

```
uint8_t oled_drawBitmap (
              uint8_t x,
              uint8_t y,
              const uint8_t picture[],
              uint8_t width,
              uint8_t height,
              uint8_t color )
```

6.18.2.10 oled_drawCircle()

```
uint8_t oled_drawCircle (
              uint8_t center_x,
              uint8_t center_y,
              uint8_t radius,
              uint8_t color )
479                                                                                 {
480     uint8_t result;
481
482     int16_t f = 1 - radius;
483     int16_t ddF_x = 1;
484     int16_t ddF_y = -2 * radius;
485     int16_t x = 0;
486     int16_t y = radius;
487
488     result = oled_drawPixel(center_x  , center_y+radius, color);
489     result = oled_drawPixel(center_x  , center_y-radius, color);
490     result = oled_drawPixel(center_x+radius, center_y  , color);
491     result = oled_drawPixel(center_x-radius, center_y  , color);
492
493     while (x<y) {
494         if (f >= 0) {
495             y--;
496             ddF_y += 2;
497             f += ddF_y;
498         }
499         x++;
500         ddF_x += 2;
```

```
501         f += ddF_x;
502
503         result = oled_drawPixel(center_x + x, center_y + y, color);
504         result = oled_drawPixel(center_x - x, center_y + y, color);
505         result = oled_drawPixel(center_x + x, center_y - y, color);
506         result = oled_drawPixel(center_x - x, center_y - y, color);
507         result = oled_drawPixel(center_x + y, center_y + x, color);
508         result = oled_drawPixel(center_x - y, center_y + x, color);
509         result = oled_drawPixel(center_x + y, center_y - x, color);
510         result = oled_drawPixel(center_x - y, center_y - x, color);
511     }
512     return result;
513 }
```

Referenced by oled_fillCircle().

Here is the call graph for this function:

Here is the caller graph for this function:

6.18.2.11 oled_drawLine()

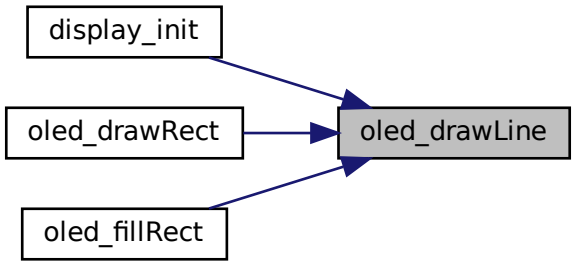
```
uint8_t oled_drawLine (
             uint8_t x1,
             uint8_t y1,
             uint8_t x2,
             uint8_t y2,
             uint8_t color )
435                                                                          {
436     uint8_t result;
437
438     int dx =  abs(x2-x1), sx = x1<x2 ?  1 :  -1;
439     int dy = -abs(y2-y1), sy = y1<y2 ?  1 :  -1;
440     int err = dx+dy, e2; /* error value e_xy */
441
442     while(1){
443         result = oled_drawPixel(x1, y1, color);
444         if (x1==x2 && y1==y2) break;
445         e2 = 2*err;
446         if (e2 > dy) { err += dy; x1 += sx; } /* e_xy+e_x > 0 */
```

```
447         if (e2 < dx) { err += dx; y1 += sy; } /* e_xy+e_y < 0 */
448     }
449
450     return result;
451 }
```

Referenced by display_init(), oled_drawRect(), and oled_fillRect().

Here is the call graph for this function:

Here is the caller graph for this function:

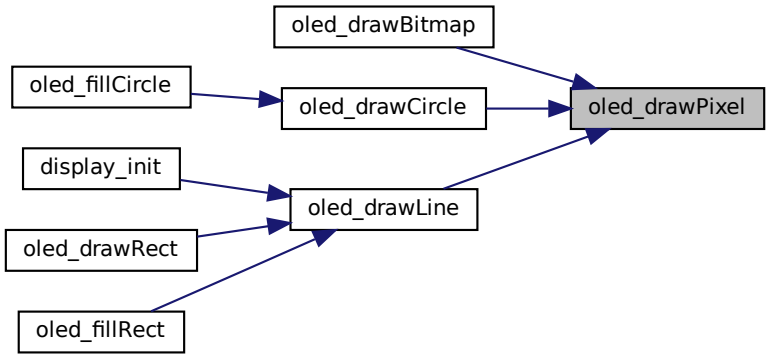


6.18.2.12 oled_drawPixel()

```
uint8_t oled_drawPixel (
              uint8_t x,
              uint8_t y,
              uint8_t color )
424                                                                 {
425     if( x > DISPLAY_WIDTH-1 || y > (DISPLAY_HEIGHT-1)) return 1; // out of Display
426
427     if( color == WHITE){
428         displayBuffer[(y / 8)][x] |= (1 « (y % 8));
429     } else {
430         displayBuffer[(y / 8)][x] &= ~(1 « (y % 8));
431     }
432
433     return 0;
434 }
```

Referenced by oled_drawBitmap(), oled_drawCircle(), and oled_drawLine().

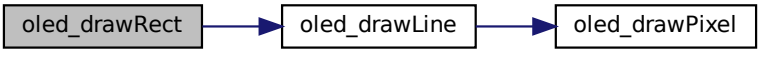
Here is the caller graph for this function:



6.18.2.13 oled_drawRect()

```
uint8_t oled_drawRect (
              uint8_t px1,
              uint8_t py1,
              uint8_t px2,
              uint8_t py2,
              uint8_t color )
452                                                                                {
453     uint8_t result;
454
455     result = oled_drawLine(px1, py1, px2, py1, color);
456     result = oled_drawLine(px2, py1, px2, py2, color);
457     result = oled_drawLine(px2, py2, px1, py2, color);
458     result = oled_drawLine(px1, py2, px1, py1, color);
459
460     return result;
461 }
```

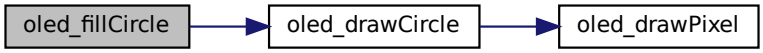
Here is the call graph for this function:



6.18.2.14 oled_fillCircle()

```
uint8_t oled_fillCircle (
              uint8_t center_x,
              uint8_t center_y,
              uint8_t radius,
              uint8_t color )
514                                                                                 {
515     uint8_t result;
516     for(uint8_t i=0; i<= radius;i++){
517         result = oled_drawCircle(center_x, center_y, i, color);
518     }
519     return result;
520 }
```

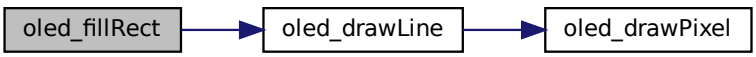
Here is the call graph for this function:



6.18.2.15 oled_fillRect()

```
uint8_t oled_fillRect (
              uint8_t px1,
              uint8_t py1,
              uint8_t px2,
              uint8_t py2,
              uint8_t color )
462                                                                             {
463     uint8_t result;
464
465     if( px1 > px2){
466         uint8_t temp = px1;
467         px1 = px2;
468         px2 = temp;
469         temp = py1;
470         py1 = py2;
471         py2 = temp;
472     }
473     for (uint8_t i=0; i<=(py2-py1); i++){
474         result = oled_drawLine(px1, py1+i, px2, py1+i, color);
475     }
476
477     return result;
478 }
```

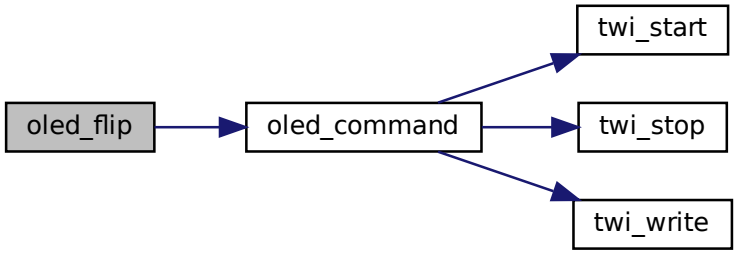
Here is the call graph for this function:



6.18.2.16 oled_flip()

```
void oled_flip (
              uint8_t flipping )
381                                {
382     uint8_t command[2] = {0xC8, 0xA1};
383     switch(flipping){
384         case 0:
385             // normal mode default at init (needs to be reload data to display)
386             command[0] = 0xC8;
387             command[1] = 0xA1;
388             oled_command(command, sizeof(command));
389             break;
390         case 1:
391             // flip horizontal && vertical (needs to be reload data to display)
392             command[0] = 0xC0;
393             command[1] = 0xA0;
394             oled_command(command, sizeof(command));
395             break;
396         case 2:
397             // flip vertical (immediate without reload data to display)
398             command[0] = 0xC0;
399             oled_command(command, sizeof(command));
400             break;
401         case 3:
402             // flip horizontal (needs to be reload data to display)
403             command[1] = 0xA0;
404             oled_command(command, sizeof(command));
405         default:
406             // do nothing
407             break;
408     }
409 }
```

Here is the call graph for this function:



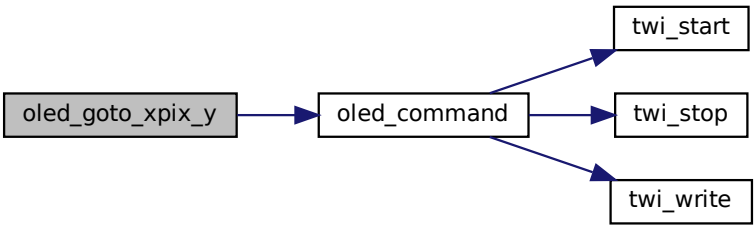
6.18.2.17 oled_goto_xpix_y()

```
void oled_goto_xpix_y (
              uint8_t x,
              uint8_t y )
174                                        {
175     if( x > (DISPLAY_WIDTH) || y > (DISPLAY_HEIGHT/8-1)) return;// out of display
176     cursorPosition.x=x;
177     cursorPosition.y=y;
178 #if defined (SSD1306) || defined (SSD1309)
179     uint8_t commandSequence[] = {0xb0+y, 0x21, x, 0x7f};
180 #elif defined SH1106
181     uint8_t commandSequence[] = {0xb0+y, 0x21, 0x00+((2+x) & (0x0f)), 0x10+( ((2+x) & (0xf0)) » 4 ),
      0x7f};
```

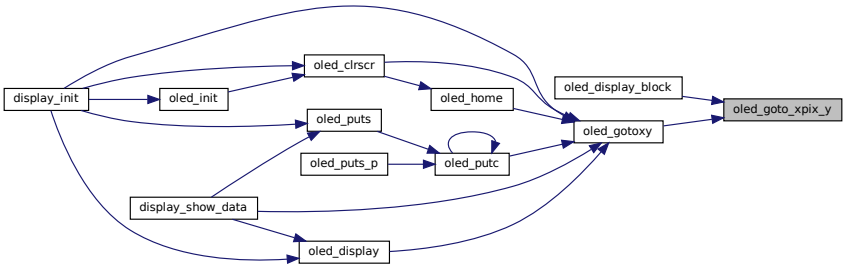
```
182 #endif
183     oled_command(commandSequence, sizeof(commandSequence));
184 }
```

Referenced by oled_display_block(), and oled_gotoxy().

Here is the call graph for this function:



Here is the caller graph for this function:

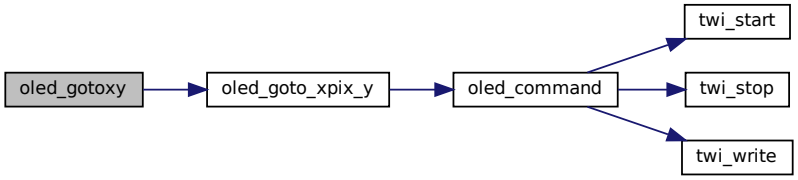


6.18.2.18 oled_gotoxy()

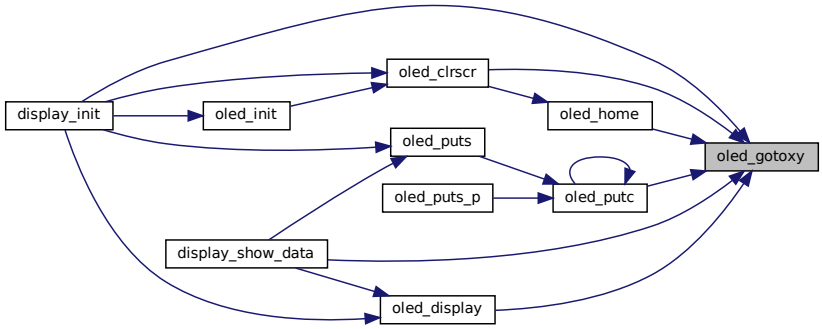
```
void oled_gotoxy (
             uint8_t x,
             uint8_t y )
170                                  {
171     x = x * sizeof(FONT[0]);
172     oled_goto_xpix_y(x,y);
173 }
```

Referenced by display_init(), display_show_data(), oled_clrscr(), oled_display(), oled_home(), and oled_putc().

Here is the call graph for this function:



Here is the caller graph for this function:



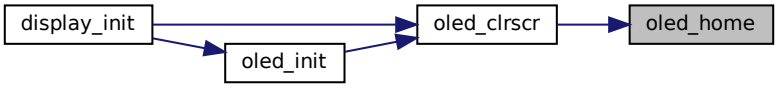
6.18.2.19 oled_home()

```
void oled_home (
            void  )
202                     {
203     oled_gotoxy(0, 0);
204 }
```

Referenced by oled_clrscr().

Here is the call graph for this function:

Here is the caller graph for this function:

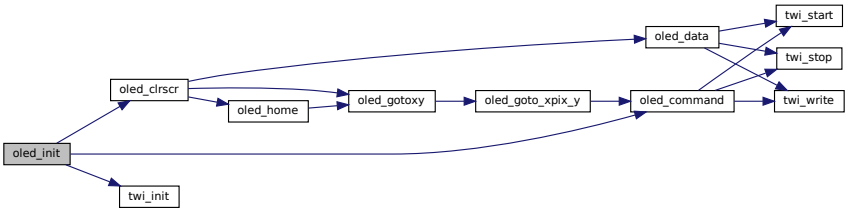


6.18.2.20 oled_init()

```
void oled_init (
             uint8_t dispAttr )
148                                 {
149 #if defined I2C
150     // i2c_init();
151     twi_init();
152 #elif defined SPI
153     DDRB |= (1 « PB2)|(1 « PB3)|(1 « PB5);
154     SPCR = (1 « SPE)|(1«MSTR)|(1«SPR0);
155     OLED_DDR |= (1 « CS_PIN)|(1 « DC_PIN)|(1 « RES_PIN);
156     OLED_PORT |= (1 « CS_PIN)|(1 « DC_PIN)|(1 « RES_PIN);
157     OLED_PORT &= ~(1 « RES_PIN);
158     _delay_ms(10);
159     OLED_PORT |= (1 « RES_PIN);
160 #endif
161
162     uint8_t commandSequence[sizeof(init_sequence)+1];
163     for (uint8_t i = 0; i < sizeof (init_sequence); i++) {
164         commandSequence[i] = (pgm_read_byte(&init_sequence[i]));
165     }
166     commandSequence[sizeof(init_sequence)]=(dispAttr);
167     oled_command(commandSequence, sizeof(commandSequence));
168     oled_clrscr();
169 }
```

Referenced by display_init().

Here is the call graph for this function:

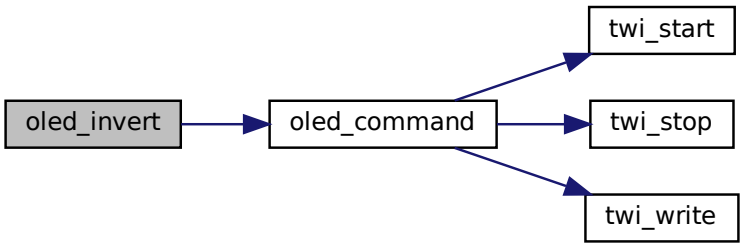


Here is the caller graph for this function:

6.18.2.21 oled_invert()

```
void oled_invert (
              uint8_t invert )
205                                {
206     uint8_t commandSequence[1];
207     if (invert != YES) {
208         commandSequence[0] = 0xA6;
209     } else {
210         commandSequence[0] = 0xA7;
211     }
212     oled_command(commandSequence, 1);
213 }
```

Here is the call graph for this function:



6.18.2.22 oled_putc()

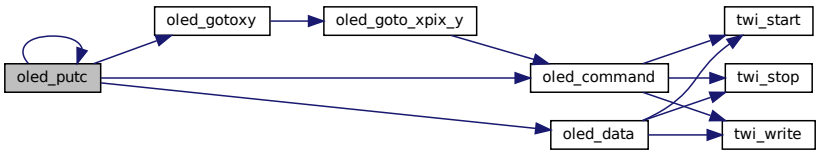
```
void oled_putc (
              char c )
227                     {
228     switch (c) {
229         case '\b':
230             // backspace
231             oled_gotoxy(cursorPosition.x-charMode, cursorPosition.y);
232             oled_putc(' ');
```

```
233             oled_gotoxy(cursorPosition.x-charMode, cursorPosition.y);
234             break;
235         case '\t':
236             // tab
237             if( (cursorPosition.x+charMode*4) < (DISPLAY_WIDTH/ sizeof(FONT[0])-charMode*4) ){
238                 oled_gotoxy(cursorPosition.x+charMode*4, cursorPosition.y);
239             }else{
240                 oled_gotoxy(DISPLAY_WIDTH/ sizeof(FONT[0]), cursorPosition.y);
241             }
242             break;
243         case '\n':
244             // linefeed
245             if(cursorPosition.y < (DISPLAY_HEIGHT/8-1)){
246                 oled_gotoxy(cursorPosition.x, cursorPosition.y+charMode);
247             }
248             break;
249         case '\r':
250             // carrige return
251             oled_gotoxy(0, cursorPosition.y);
252             break;
253         default:
254             // char doesn't fit in line
255             if( (cursorPosition.x >= DISPLAY_WIDTH-sizeof(FONT[0])) || (c < ' ') ) break;
256             // mapping char
257             c -= ' ';
258             if (c >= pgm_read_byte(&special_char[0][1]) ) {
259                 char temp = c;
260                 c = 0xff;
261                 for (uint8_t i=0; pgm_read_byte(&special_char[i][1]) != 0xff; i++) {
262                     if ( pgm_read_byte(&special_char[i][0])-' ' == temp ) {
263                         c = pgm_read_byte(&special_char[i][1]);
264                         break;
265                     }
266                 }
267                 if ( c == 0xff ) break;
268             }
269             // print char at display
270 #ifdef GRAPHICMODE
271             if (charMode == DOUBLESIZE) {
272                 uint16_t doubleChar[sizeof(FONT[0])];
273                 uint8_t dChar;
274                 if ((cursorPosition.x+2*sizeof(FONT[0]))>DISPLAY_WIDTH) break;
275 
276                 for (uint8_t i=0; i < sizeof(FONT[0]); i++) {
277                     doubleChar[i] = 0;
278                     dChar = pgm_read_byte(&(FONT[(uint8_t)c][i]));
279                     for (uint8_t j=0; j<8; j++) {
280                         if ((dChar & (1 << j))) {
281                             doubleChar[i] |= (1 << (j*2));
282                             doubleChar[i] |= (1 << ((j*2)+1));
283                         }
284                     }
285                 }
286                 for (uint8_t i = 0; i < sizeof(FONT[0]); i++)
287                 {
288                     // load bit-pattern from flash
289                     displayBuffer[cursorPosition.y+1][cursorPosition.x+(2*i)] = doubleChar[i] >> 8;
290                     displayBuffer[cursorPosition.y+1][cursorPosition.x+(2*i)+1] = doubleChar[i] >> 8;
291                     displayBuffer[cursorPosition.y][cursorPosition.x+(2*i)] = doubleChar[i] & 0xff;
292                     displayBuffer[cursorPosition.y][cursorPosition.x+(2*i)+1] = doubleChar[i] & 0xff;
293                 }
294                 cursorPosition.x += sizeof(FONT[0])*2;
295             } else {
296                 if ((cursorPosition.x+sizeof(FONT[0]))>DISPLAY_WIDTH) break;
297 
298                 for (uint8_t i = 0; i < sizeof(FONT[0]); i++)
299                 {
300                     // load bit-pattern from flash
301                     displayBuffer[cursorPosition.y][cursorPosition.x+i]
    =pgm_read_byte(&(FONT[(uint8_t)c][i]));
302                 }
303                 cursorPosition.x += sizeof(FONT[0]);
304             }
305 #elif defined TEXTMODE
306             if (charMode == DOUBLESIZE) {
307                 uint16_t doubleChar[sizeof(FONT[0])];
308                 uint8_t dChar;
309                 if ((cursorPosition.x+2*sizeof(FONT[0]))>DISPLAY_WIDTH) break;
310 
311                 for (uint8_t i=0; i < sizeof(FONT[0]); i++) {
312                     doubleChar[i] = 0;
313                     dChar = pgm_read_byte(&(FONT[(uint8_t)c][i]));
314                     for (uint8_t j=0; j<8; j++) {
315                         if ((dChar & (1 << j))) {
316                             doubleChar[i] |= (1 << (j*2));
317                             doubleChar[i] |= (1 << ((j*2)+1));
318                         }
```

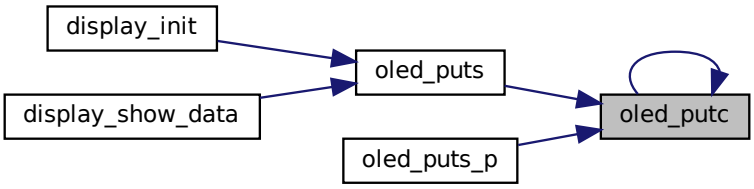
```
319                     }
320                 }
321                 uint8_t data[sizeof(FONT[0])*2];
322                 for (uint8_t i = 0; i < sizeof(FONT[0]); i++)
323                 {
324                     // print font to ram, print 6 columns
325                     data[i<<1]=(doubleChar[i] & 0xff);
326                     data[(i<<1)+1]=(doubleChar[i] & 0xff);
327                 }
328                 oled_data(data, sizeof(FONT[0])*2);
329 
330 #if defined (SSD1306) || defined (SSD1309)
331                 uint8_t commandSequence[] = {0xb0+cursorPosition.y+1,
332                     0x21,
333                     cursorPosition.x,
334                     0x7f};
335 #elif defined SH1106
336                 uint8_t commandSequence[] = {0xb0+cursorPosition.y+1,
337                     0x21,
338                     0x00+((2+cursorPosition.x) & (0x0f)),
339                     0x10+( ((2+cursorPosition.x) & (0xf0)) >> 4 ),
340                     0x7f};
341 #endif
342                 oled_command(commandSequence, sizeof(commandSequence));
343 
344                 for (uint8_t i = 0; i < sizeof(FONT[0]); i++)
345                 {
346                     // print font to ram, print 6 columns
347                     data[i<<1]=(doubleChar[i] >> 8);
348                     data[(i<<1)+1]=(doubleChar[i] >> 8);
349                 }
350                 oled_data(data, sizeof(FONT[0])*2);
351 
352                 commandSequence[0] = 0xb0+cursorPosition.y;
353 #if defined (SSD1306) || defined (SSD1309)
354                 commandSequence[2] = cursorPosition.x+(2*sizeof(FONT[0]));
355 #elif defined SH1106
356                 commandSequence[2] = 0x00+((2+cursorPosition.x+(2*sizeof(FONT[0]))) & (0x0f));
357                 commandSequence[3] = 0x10+( ((2+cursorPosition.x+(2*sizeof(FONT[0]))) & (0xf0)) >> 4 );
358 #endif
359                 oled_command(commandSequence, sizeof(commandSequence));
360                 cursorPosition.x += sizeof(FONT[0])*2;
361             } else {
362                 uint8_t data[sizeof(FONT[0])];
363                 if ((cursorPosition.x+sizeof(FONT[0]))>DISPLAY_WIDTH) break;
364 
365                 for (uint8_t i = 0; i < sizeof(FONT[0]); i++)
366                 {
367                     // print font to ram, print 6 columns
368                     data[i]=(pgm_read_byte(&(FONT[(uint8_t)c][i])));
369                 }
370                 oled_data(data, sizeof(FONT[0]));
371                 cursorPosition.x += sizeof(FONT[0]);
372             }
373 #endif
374             break;
375     }
376 
377 }
```

Referenced by oled_putc(), oled_puts(), and oled_puts_p().

Here is the call graph for this function:



Here is the caller graph for this function:

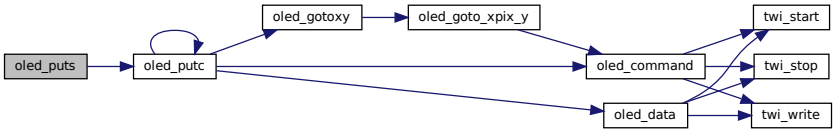


6.18.2.23 oled_puts()

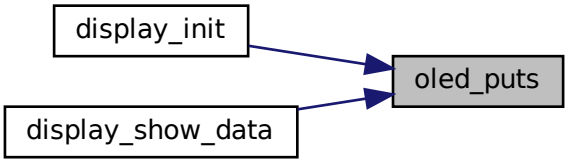
```
void oled_puts (
              const char * s )
410                                 {
411     while (*s) {
412         oled_putc(*s++);
413     }
414 }
```

Referenced by display_init(), and display_show_data().

Here is the call graph for this function:



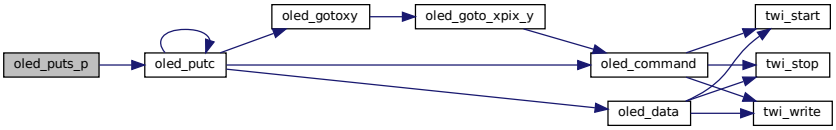
Here is the caller graph for this function:



6.18.2.24 oled_puts_p()

```
void oled_puts_p (
             const char * progmem_s )
415                                      {
416     register uint8_t c;
417     while ((c = pgm_read_byte(progmem_s++))) {
418         oled_putc(c);
419     }
420 }
```

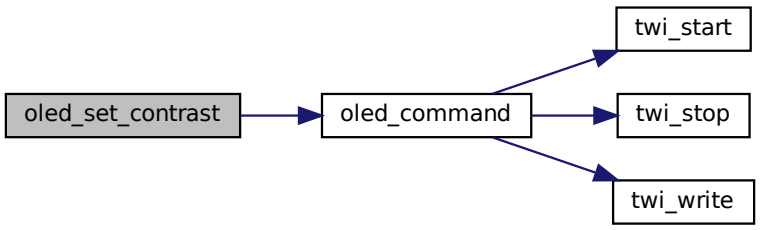
Here is the call graph for this function:



6.18.2.25 oled_set_contrast()

```
void oled_set_contrast (
             uint8_t contrast )
223                                       {
224     uint8_t commandSequence[2] = {0x81, contrast};
225     oled_command(commandSequence, sizeof(commandSequence));
226 }
```

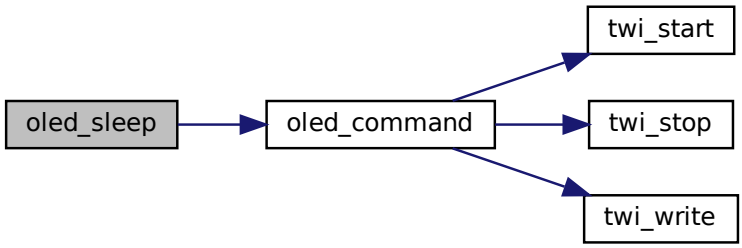
Here is the call graph for this function:



6.18.2.26 oled_sleep()

```
void oled_sleep (
             uint8_t sleep )
214                             {
215     uint8_t commandSequence[1];
216     if (sleep != YES) {
217         commandSequence[0] = 0xAF;
218     } else {
219         commandSequence[0] = 0xAE;
220     }
221     oled_command(commandSequence, 1);
222 }
```

Here is the call graph for this function:



6.19 oled.h

Go to the documentation of this file.

```
1 /*
2 * This file is part of lcd library for ssd1306/ssd1309/sh1106 oled-display.
3 *
4 * lcd library for ssd1306/ssd1309/sh1106 oled-display is free software:  you can redistribute it and/or
    modify
5 * it under the terms of the GNU General Public License as published by
6 * the Free Software Foundation, either version 3 of the License, or any later version.
7 *
8 * lcd library for ssd1306/ssd1309/sh1106 oled-display is distributed in the hope that it will be useful,
9 * but WITHOUT ANY WARRANTY; without even the implied warranty of
10 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
11 * GNU General Public License for more details.
12 *
13 * You should have received a copy of the GNU General Public License
14 * along with Foobar.    If not, see <http://www.gnu.org/licenses/>.
15 *
16 * Diese Datei ist Teil von lcd library for ssd1306/ssd1309/sh1106 oled-display.
17 *
18 * lcd library for ssd1306/ssd1309/sh1106 oled-display ist Freie Software:  Sie können es unter den
    Bedingungen
19 * der GNU General Public License, wie von der Free Software Foundation,
20 * Version 3 der Lizenz oder jeder späteren
21 * veröffentlichten Version, weiterverbreiten und/oder modifizieren.
22 *
23 * lcd library for ssd1306/ssd1309/sh1106 oled-display wird in der Hoffnung, dass es nützlich sein wird,
    aber
24 * OHNE JEDE GEWÄHRLEISTUNG, bereitgestellt; sogar ohne die implizite
25 * Gewährleistung der MARKTFÄHIGKEIT oder EIGNUNG FÜR EINEN BESTIMMTEN ZWECK.
26 * Siehe die GNU General Public License für weitere Details.
27 *
28 * Sie sollten eine Kopie der GNU General Public License zusammen mit diesem
29 * Programm erhalten haben.  Wenn nicht, siehe <http://www.gnu.org/licenses/>.
30 *
31 *  lcd.h
32 *
```

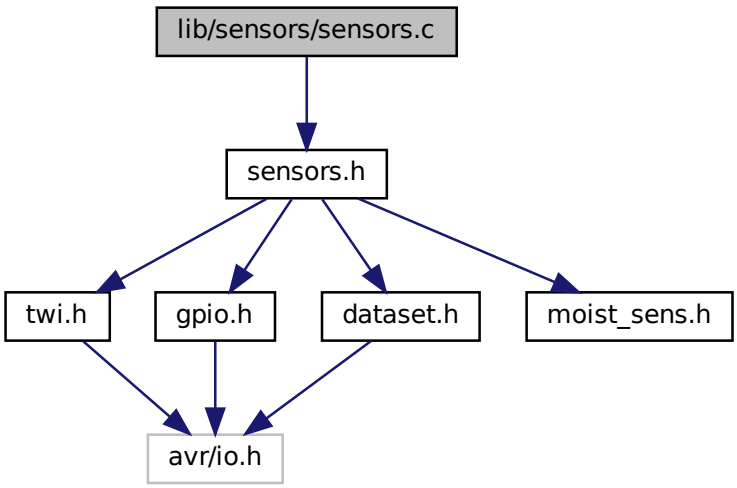
```
33 *  Created by Michael Köhler on 22.12.16.
34 *  Copyright 2016 Skie-Systems.  All rights reserved.
35 *
36 *  lib for OLED-Display with ssd1306/ssd1309/sh1106-Controller
37 *  first dev-version only for I2C-Connection
38 *  at ATMega328P like Arduino Uno
39 *
40 *  at GRAPHICMODE lib needs SRAM for display
41 *  DISPLAY-WIDTH * DISPLAY-HEIGHT + 2 bytes
42 */
43
44 #ifndef OLED_H
45 #define OLED_H
46
47 #ifdef __cplusplus
48 extern "C" {
49 #endif
50
51 #if (__GNUC__ * 100 + __GNUC_MINOR__) < 303
52 # error "This library requires AVR-GCC 3.3 or later, update to newer AVR-GCC compiler !"
53 #endif
54
55 #include <inttypes.h>
56 #include <avr/pgmspace.h>
57
58     /* TODO: define bus */
59 #define I2C  // I2C or SPI
60     /* TODO: define displaycontroller */
61 #define SH1106  // or SSD1306, check datasheet of your display
62     /* TODO: define displaymode */
63 #define GRAPHICMODE  // for text and graphic
64     // TEXTMODE // for only text to display,
65     /* TODO: define font */
66 #define FONT  ssd1306oled_font  // Refer font-name at font.h
67
68     // using 7-bit-adress for lcd-library
69     // if you use your own library for twi check I2C-adress-handle
70 #define OLED_I2C_ADR (0x3c)  // 7 bit slave-adress without r/w-bit
71     // e.g.  8 bit slave-adress:
72     // 0x78 = adress 0x3C with cleared r/w-bit (write-mode)
73
74
75 #ifdef I2C
76 // # include "i2c.h"
77 # include "twi.h"
78 #elif defined SPI
79 // If you want to use your other lib/function for SPI replace SPI-commands
80 # define OLED_PORT PORTB
81 # define OLED_DDR  DDRB
82 # define RES_PIN  PB0
83 # define DC_PIN   PB1
84 # define CS_PIN   PB2
85 #endif
86
87 #ifndef YES
88 # define YES 1
89 #endif
90
91 #define NORMALSIZE 1
92 #define DOUBLESIZE 2
93
94 #define OLED_DISP_OFF 0xAE
95 #define OLED_DISP_ON 0xAF
96
97 #define WHITE 0x01
98 #define BLACK 0x00
99
100 #define DISPLAY_WIDTH 128
101 #define DISPLAY_HEIGHT 64
102
103 // Transmit command or data to display
104 void oled_command(uint8_t cmd[], uint8_t size);
105 void oled_data(uint8_t data[], uint16_t size);
106 void oled_init(uint8_t dispAttr);
107 void oled_home(void);  // set cursor to 0,0
108 void oled_invert(uint8_t invert);  // invert display
109 void oled_sleep(uint8_t sleep);    // display goto sleep (power off)
110 void oled_set_contrast(uint8_t contrast);    // set contrast for display
111 void oled_puts(const char* s);             // print string, \n-terminated, from ram on screen (TEXTMODE)
112                       // or buffer (GRAPHICMODE)
113 void oled_puts_p(const char* progmem_s);  // print string from flash on screen (TEXTMODE)
114 // or buffer (GRAPHICMODE)
115
116 void oled_clrscr(void);  // clear screen (and buffer at GRFAICMODE)
117 void oled_gotoxy(uint8_t x, uint8_t y);  // set curser at pos x, y.  x means character,
118 // y means line (page, refer lcd manual)
119 void oled_goto_xpix_y(uint8_t x, uint8_t y); // set curser at pos x, y.  x means pixel,
```

```
120 // y means line (page, refer lcd manual)
121 void oled_putc(char c);  // print character on screen at TEXTMODE
122 // at GRAPHICMODE print character to buffer
123 void oled_charMode(uint8_t mode);  // set size of chars
124 void oled_flip(uint8_t flipping);  // flip display,
125                     // flipping == 0:  no flip (normal mode)
126                         // == 1:  flip horizontal & vertical
127                         // == 2:  flip(mirrored) vertical
128                         // == 3:  flip(mirrored) horizontal
129 #if defined GRAPHICMODE
130     uint8_t oled_drawPixel(uint8_t x, uint8_t y, uint8_t color);
131     uint8_t oled_drawLine(uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, uint8_t color);
132     uint8_t oled_drawRect(uint8_t px1, uint8_t py1, uint8_t px2, uint8_t py2, uint8_t color);
133     uint8_t oled_fillRect(uint8_t px1, uint8_t py1, uint8_t px2, uint8_t py2, uint8_t color);
134     uint8_t oled_drawCircle(uint8_t center_x, uint8_t center_y, uint8_t radius, uint8_t color);
135     uint8_t oled_fillCircle(uint8_t center_x, uint8_t center_y, uint8_t radius, uint8_t color);
136     uint8_t oled_drawBitmap(uint8_t x, uint8_t y, const uint8_t picture[], uint8_t width, uint8_t
      height, uint8_t color);
137     void oled_display(void);       // copy buffer to display RAM
138     void oled_clear_buffer(void);  // clear display buffer
139     uint8_t oled_check_buffer(uint8_t x, uint8_t y); // read a pixel value from the display buffer
140     void oled_display_block(uint8_t x, uint8_t line, uint8_t width); // display (part of) a display line
141 #endif
142
143 #ifdef __cplusplus
144 }
145 #endif
146
147 #endif /*  OLED_H  */
```

6.20 lib/sensors/sensors.c File Reference

```
#include "sensors.h"
```

Include dependency graph for sensors.c:



Functions

- void sensors_init ()
- void sensors_update_dataset (dataset_t *data)

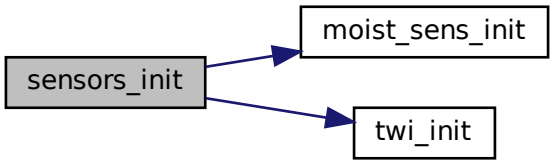
6.20.1 Function Documentation

6.20.1.1 sensors_init()

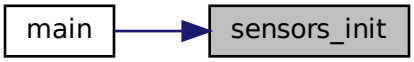
```
void sensors_init ( )
4 {
5     twi_init();
6     moist_sens_init();
7 }
```

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



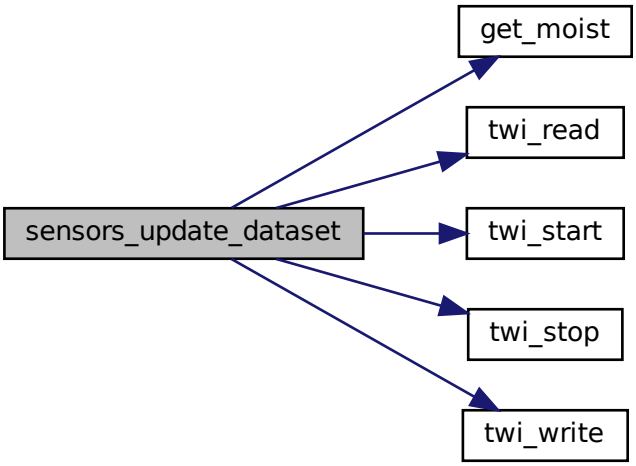
6.20.1.2 sensors_update_dataset()

```
void sensors_update_dataset (
              dataset_t * data )
11 {
12     twi_start();
13     if (twi_write((DHT22_ADDR«1) | TWI_WRITE) == 0)
14     {
15         // Set internal memory location
16         twi_write(SENSOR_HUM_MEM);
17         twi_stop();
18
19         // Read data from internal memory
```

```
20         twi_start();
21         twi_write((DHT22_ADDR<<1) | TWI_READ);
22         data->hum = twi_read(TWI_ACK);
23         twi_read(TWI_ACK); // skip hum decimal
24         data->temp = twi_read(TWI_ACK);
25         twi_read(TWI_ACK); // skip temp decimal
26     }
27     twi_stop();
28
29     data->moist = get_moist();
30 }
```

Referenced by main().

Here is the call graph for this function:



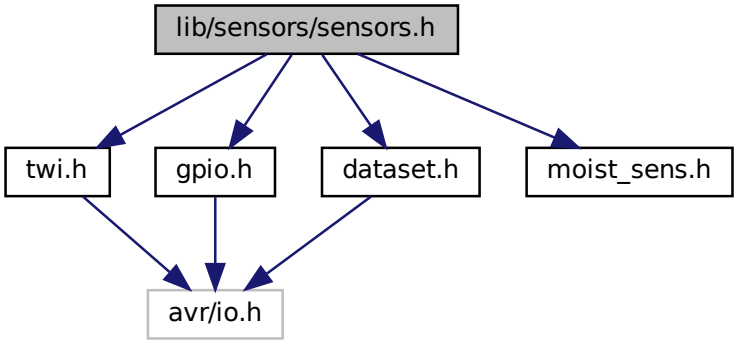
Here is the caller graph for this function:

6.21 lib/sensors/sensors.h File Reference

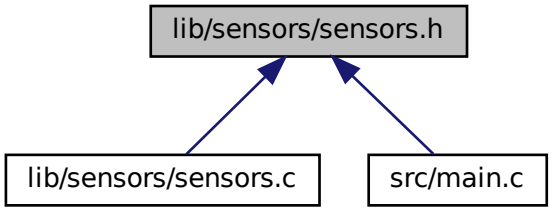
```
#include <twi.h>
#include <gpio.h>
#include <dataset.h>
```

```
#include <moist_sens.h>
```

Include dependency graph for sensors.h:



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



Macros

- #define DHT22_ADDR 0x5c
- #define SENSOR_HUM_MEM 0
- #define SENSOR_TEMP_MEM 2
- #define SENSOR_CHECKSUM 4

Functions

- void sensors_init ()
- void sensors_update_dataset (dataset_t *data)

6.21.1 Macro Definition Documentation

6.21.1.1 DHT22_ADDR

```
#define DHT22_ADDR 0x5c
```

6.21.1.2 SENSOR_CHECKSUM

```
#define SENSOR_CHECKSUM 4
```

6.21.1.3 SENSOR_HUM_MEM

```
#define SENSOR_HUM_MEM 0
```

6.21.1.4 SENSOR_TEMP_MEM

```
#define SENSOR_TEMP_MEM 2
```

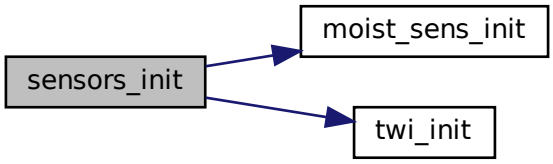
6.21.2 Function Documentation

6.21.2.1 sensors_init()

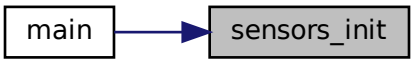
```
void sensors_init ( )
4 {
5     twi_init();
6     moist_sens_init();
7 }
```

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

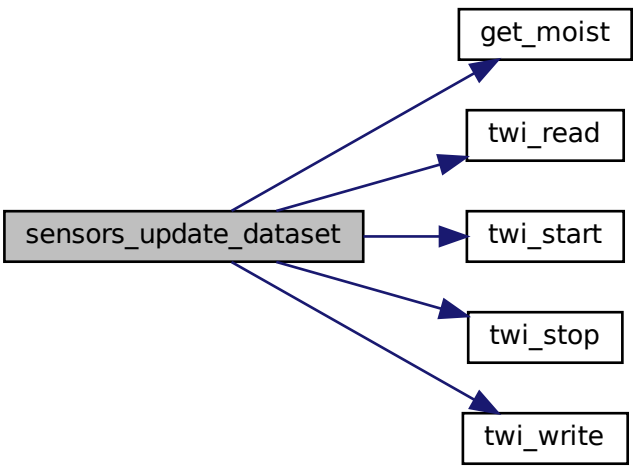


6.21.2.2 sensors_update_dataset()

```
void sensors_update_dataset (
              dataset_t * data )
11 {
12     twi_start();
13     if (twi_write((DHT22_ADDR«1) | TWI_WRITE) == 0)
14     {
15         // Set internal memory location
16         twi_write(SENSOR_HUM_MEM);
17         twi_stop();
18 
19         // Read data from internal memory
20         twi_start();
21         twi_write((DHT22_ADDR«1) | TWI_READ);
22         data->hum = twi_read(TWI_ACK);
23         twi_read(TWI_ACK); // skip hum decimal
24         data->temp = twi_read(TWI_ACK);
25         twi_read(TWI_ACK); // skip temp decimal
26     }
27     twi_stop();
28 
29     data->moist = get_moist();
30 }
```

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

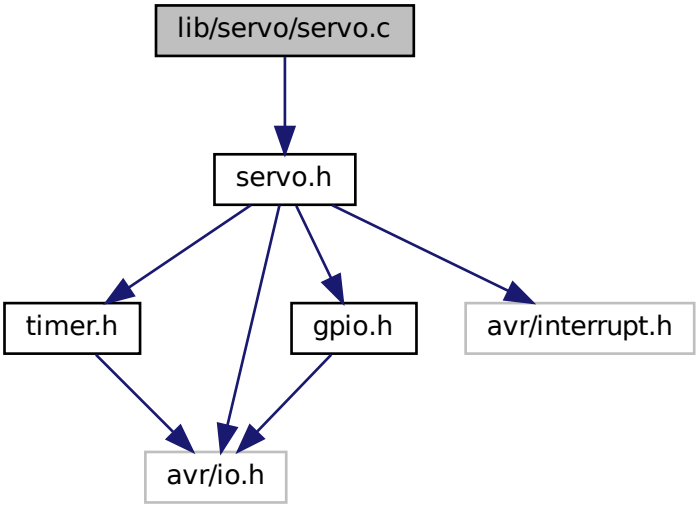
6.22 sensors.h

Go to the documentation of this file.

```
1 #ifndef SENSORS_H
2 #define SENSORS_H
3
4 #include <twi.h>
5 #include <gpio.h>
6 #include <dataset.h>
7 #include <moist_sens.h>
8
9 #define DHT22_ADDR 0x5c
10 #define SENSOR_HUM_MEM 0
11 #define SENSOR_TEMP_MEM 2
12 #define SENSOR_CHECKSUM 4
13
14
15 /*************************************************
16 *  @brief  Inicialization of all sensors
17 *  @param  None
18 *  @return None
19 *************************************************/
20 void sensors_init();
21
22
23 /*************************************************
24 *  @brief  Updates dataset with current
25 *          data from sensors
26 *  @param  dataset Actual measured data
27 *  @return None
28 *************************************************/
29 void sensors_update_dataset(dataset_t *data);
30
31 #endif
32
```

6.23 lib/servo/servo.c File Reference

`#include "servo.h"`
Include dependency graph for servo.c:



Functions

- void servo_init (servo_t *servo, volatile uint8_t *reg, uint8_t pin)
- void servo_set_value (servo_t *servo, uint8_t value)
- void servo_50us_interrupt_handler (servo_t *servo)

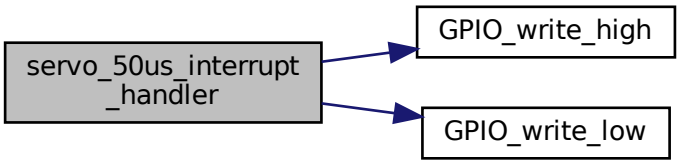
6.23.1 Function Documentation

6.23.1.1 servo_50us_interrupt_handler()

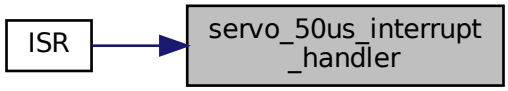
```
void servo_50us_interrupt_handler (
            servo_t * servo )
22 {
23     static uint16_t tick = 0;
24     if (tick == servo->value/6+14) GPIO_write_low(servo->reg, servo->pin);
25     if (tick >= 387)
26     {
27         //PORTB |= (1 « 0);
28         GPIO_write_high(servo->reg, servo->pin);
29         tick=0;
30     }
31     else tick++;
32 }
```

Referenced by ISR().

Here is the call graph for this function:



Here is the caller graph for this function:

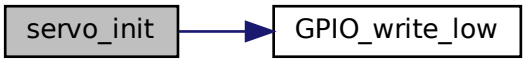


6.23.1.2 servo_init()

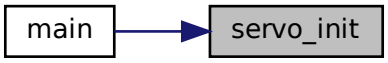
```
void servo_init (
              servo_t * servo,
              volatile uint8_t * reg,
              uint8_t pin )
4 {
5     servo->reg = reg;
6     servo->pin = pin;
7
8     TIM0_OVF_128US();
9     TIM0_OVF_ENABLE();
10     sei();
11
12     //GPIO_mode_output(servo->reg, servo->pin);
13     GPIO_write_low(servo->reg, servo->pin);
14 }
```

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

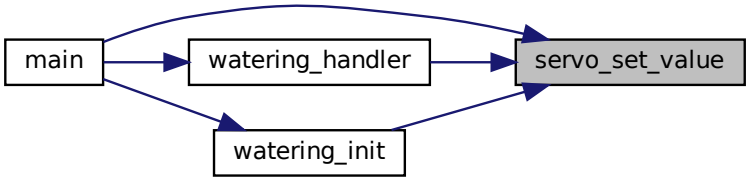


6.23.1.3 servo_set_value()

```
void servo_set_value (
             servo_t * servo,
             uint8_t value )
17 {
18     servo->value = value;
19 }
```

Referenced by main(), watering_handler(), and watering_init().

Here is the caller graph for this function:

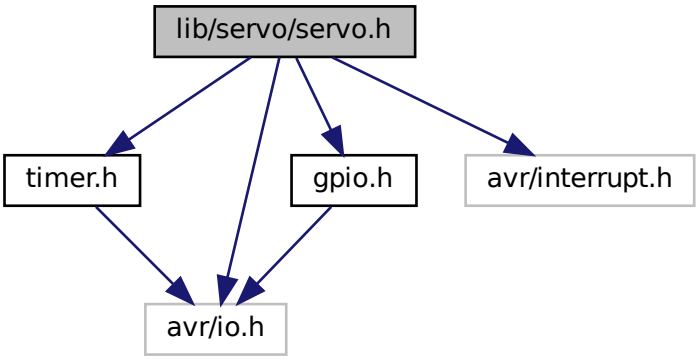


6.24 lib/servo/servo.h File Reference

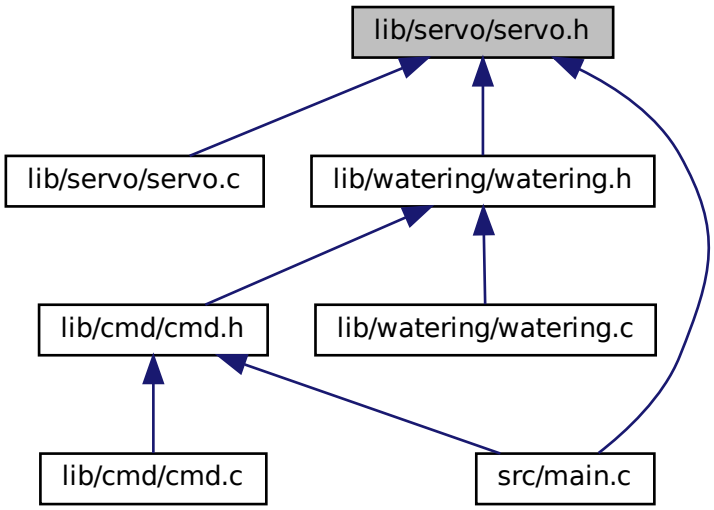
```
#include <timer.h>
#include <gpio.h>
#include <avr/io.h>
```

```
#include <avr/interrupt.h>
```

Include dependency graph for servo.h:



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



Classes

- struct servo_t

Functions

- void servo_init (servo_t *servo, volatile uint8_t *reg, uint8_t pin)
- void servo_set_value (servo_t *servo, uint8_t value)
- void servo_50us_interrupt_handler (servo_t *servo)

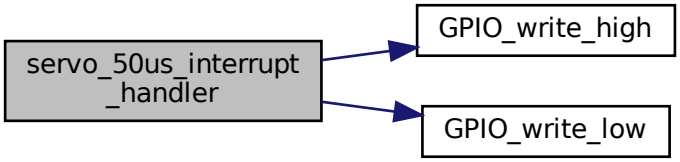
6.24.1 Function Documentation

6.24.1.1 servo_50us_interrupt_handler()

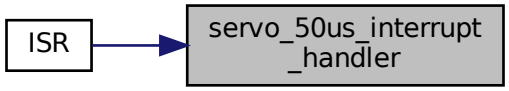
```
void servo_50us_interrupt_handler (
              servo_t * servo )
22 {
23     static uint16_t tick = 0;
24     if (tick == servo->value/6+14) GPIO_write_low(servo->reg, servo->pin);
25     if (tick >= 387)
26     {
27         //PORTB |= (1 « 0);
28         GPIO_write_high(servo->reg, servo->pin);
29         tick=0;
30     }
31     else tick++;
32 }
```

Referenced by ISR().

Here is the call graph for this function:



Here is the caller graph for this function:

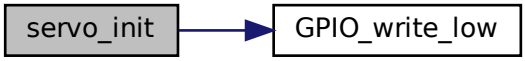


6.24.1.2 servo_init()

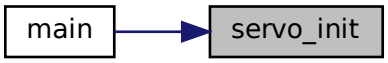
```
void servo_init (
              servo_t * servo,
              volatile uint8_t * reg,
              uint8_t pin )
4 {
5     servo->reg = reg;
6     servo->pin = pin;
7
8     TIM0_OVF_128US();
9     TIM0_OVF_ENABLE();
10     sei();
11
12     //GPIO_mode_output(servo->reg, servo->pin);
13     GPIO_write_low(servo->reg, servo->pin);
14 }
```

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

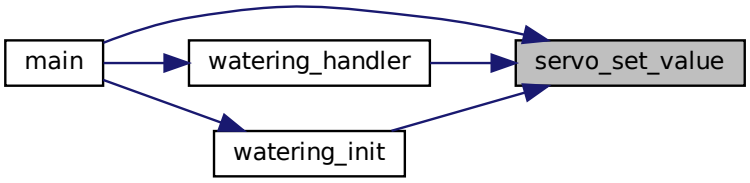


6.24.1.3 servo_set_value()

```
void servo_set_value (
              servo_t * servo,
              uint8_t value )
17 {
18     servo->value = value;
19 }
```

Referenced by main(), watering_handler(), and watering_init().

Here is the caller graph for this function:



6.25 servo.h

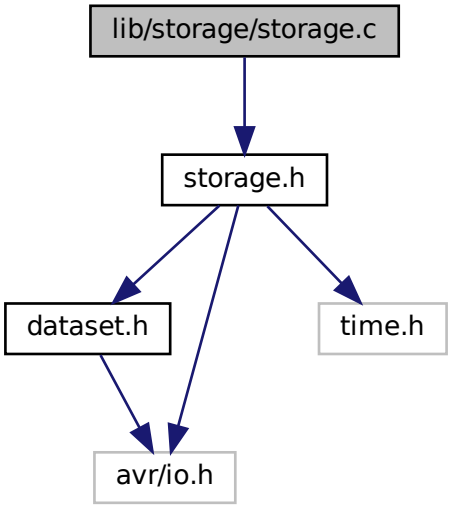
Go to the documentation of this file.

```
1 #ifndef SERVO_H
2 #define SERVO_H
3
4 #include <timer.h>
5 #include <gpio.h>
6 #include <avr/io.h>
7 #include <avr/interrupt.h>
8
19 typedef struct
20 {
21     volatile uint8_t *reg;
22     uint8_t pin;
23     uint8_t value;
24 } servo_t;
25
26
27 /*************************************************
28 *  @brief  Servo inicialization
29 *  @param  servo ---
30 *  @param  reg ---
31 *  @param  pin ---
32 *  @return None
33 *************************************************/
34 void servo_init(servo_t *servo, volatile uint8_t *reg, uint8_t pin);
35
36 /*************************************************
37 *  @brief  ---
38 *  @param  servo ---
39 *  @param  value ---
40 *  @return None
41 *************************************************/
42 void servo_set_value(servo_t *servo, uint8_t value);
43
44 /*************************************************
45 *  @brief  ---
46 *  @param  servo ---
47 *  @return None
48 *************************************************/
49 void servo_50us_interrupt_handler(servo_t *servo);
50
51 #endif
52
```

6.26 lib/storage/storage.c File Reference

```
#include "storage.h"
```

Include dependency graph for storage.c:



Functions

- void EEPROM_write (uint16_t addr, uint8_t val)
- uint8_t EEPROM_read (uint16_t addr)
- void storage_read (storage_t *storage, dataset_t *data, uint8_t pos)
- void storage_write (storage_t *storage, dataset_t *data)
- void storage_init (storage_t *storage)

6.26.1 Function Documentation

6.26.1.1 EEPROM_read()

```
uint8_t EEPROM_read (
              uint16_t addr )
20 {
21     /* Wait for completion of previous write */
22     while(EECR & (1<<EEPE));
23     /* Set up address register */
24     EEAR = addr;
25     /* Start eeprom read by writing EERE */
26     EECR |= (1<<EERE);
27     /* Return data from Data Register */
28     return EEDR;
29 }
```

Referenced by storage_read().

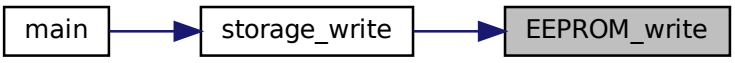
Here is the caller graph for this function:

6.26.1.2 EEPROM_write()

```
void EEPROM_write (
              uint16_t addr,
              uint8_t val )
6 {
7     /* Wait for completion of previous write */
8     while(EECR & (1<<EEPE)) ;
9     /* Set up address and Data Registers */
10     EEAR = addr;
11     EEDR = val;
12     /* Write logical one to EEMPE */
13     EECR |= (1<<EEMPE);
14     /* Start eeprom write by setting EEPE */
15     EECR |= (1<<EEPE);
16 }
```

Referenced by storage_write().

Here is the caller graph for this function:

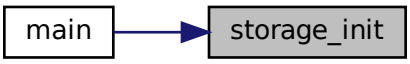


6.26.1.3 storage_init()

```
void storage_init (
              storage_t * storage )
60 {
61     storage->buffer_start = 0;
62 }
```

Referenced by main().

Here is the caller graph for this function:



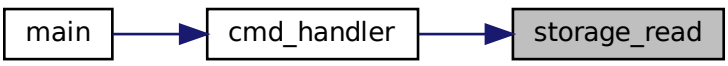
6.26.1.4 storage_read()

```
void storage_read (
              storage_t * storage,
              dataset_t * data,
              uint8_t pos )
33 {
34     uint8_t *iter = ((uint8_t *)data);
35     while(pos > storage->buffer_start)
36         pos -= 1000 / sizeof(dataset_t);
37
38     for(uint8_t i = 0; i < sizeof(dataset_t); i++)
39     {
40         iter[i] = EEPROM_read(i + storage->buffer_start - pos*sizeof(dataset_t));
41     }
42 }
```

Referenced by cmd_handler().

Here is the call graph for this function:

Here is the caller graph for this function:



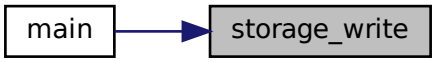
6.26.1.5 storage_write()

```
void storage_write (
              storage_t * storage,
              dataset_t * data )
45 {
46     uint8_t *iter = ((uint8_t *)data);
47     for(uint8_t i = 0; i < sizeof(dataset_t); i++)
48     {
49         EEPROM_write(i + storage->buffer_start, iter[i]);
50     }
51     storage->buffer_start += sizeof(dataset_t);
52
53     if(storage->buffer_start + sizeof(dataset_t) > 1000)
54     {
55         storage->buffer_start = 0;
56     }
57 }
```

Referenced by main().

Here is the call graph for this function:

Here is the caller graph for this function:

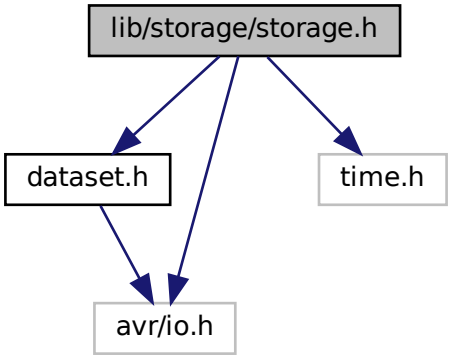


6.27 lib/storage/storage.h File Reference

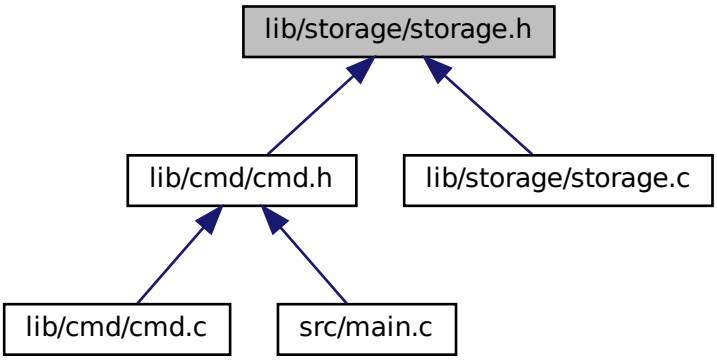
```
#include <dataset.h>
#include <avr/io.h>
```

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

Include dependency graph for storage.h:



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



Classes

- struct storage_t

Functions

- void storage_init (storage_t *storage)
- void EEPROM_write (uint16_t addr, uint8_t val)
- uint8_t EEPROM_read (uint16_t addr)
- void storage_write (storage_t *storage, dataset_t *data)
- void storage_read (storage_t *storage, dataset_t *data, uint8_t pos)

6.27.1 Function Documentation

6.27.1.1 EEPROM_read()

```
uint8_t EEPROM_read (
              uint16_t addr )
20 {
21     /* Wait for completion of previous write */
22     while(EECR & (1<<EEPE));
23     /* Set up address register */
24     EEAR = addr;
25     /* Start eeprom read by writing EERE */
26     EECR |= (1<<EERE);
27     /* Return data from Data Register */
28     return EEDR;
29 }
```

Referenced by storage_read().

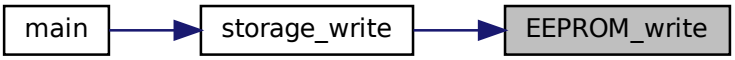
Here is the caller graph for this function:

6.27.1.2 EEPROM_write()

```
void EEPROM_write (
              uint16_t addr,
              uint8_t val )
6 {
7     /* Wait for completion of previous write */
8     while(EECR & (1<<EEPE)) ;
9     /* Set up address and Data Registers */
10     EEAR = addr;
11     EEDR = val;
12     /* Write logical one to EEMPE */
13     EECR |= (1<<EEMPE);
14     /* Start eeprom write by setting EEPE */
15     EECR |= (1<<EEPE);
16 }
```

Referenced by storage_write().

Here is the caller graph for this function:

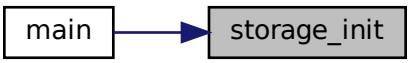


6.27.1.3 storage_init()

```
void storage_init (
              storage_t * storage )
60 {
61     storage->buffer_start = 0;
62 }
```

Referenced by main().

Here is the caller graph for this function:



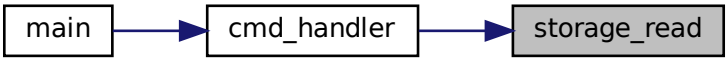
6.27.1.4 storage_read()

```
void storage_read (
              storage_t * storage,
              dataset_t * data,
              uint8_t pos )
33 {
34     uint8_t *iter = ((uint8_t *)data);
35     while(pos > storage->buffer_start)
36         pos -= 1000 / sizeof(dataset_t);
37
38     for(uint8_t i = 0; i < sizeof(dataset_t); i++)
39     {
40         iter[i] = EEPROM_read(i + storage->buffer_start - pos*sizeof(dataset_t));
41     }
42 }
```

Referenced by cmd_handler().

Here is the call graph for this function:

Here is the caller graph for this function:



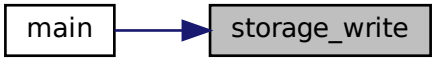
6.27.1.5 storage_write()

```
void storage_write (
            storage_t * storage,
            dataset_t * data )
45 {
46     uint8_t *iter = ((uint8_t *)data);
47     for(uint8_t i = 0; i < sizeof(dataset_t); i++)
48     {
49         EEPROM_write(i + storage->buffer_start, iter[i]);
50     }
51     storage->buffer_start += sizeof(dataset_t);
52 
53     if(storage->buffer_start + sizeof(dataset_t) > 1000)
54     {
55         storage->buffer_start = 0;
56     }
57 }
```

Referenced by main().

Here is the call graph for this function:

Here is the caller graph for this function:



6.28 storage.h

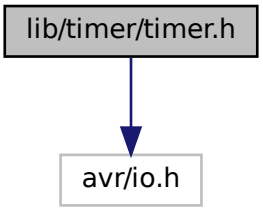
Go to the documentation of this file.

```
1 #ifndef STORAGE_H
2 #define STORAGE_H
3
4 #include <dataset.h>
5 #include <avr/io.h>
6 #include <time.h>
7
8 typedef struct {
9     uint16_t buffer_start;
10 } storage_t;
11
12 /*****************************************************
13 *  @brief  Storage inicialization
14 *  @param  storage Storage setup
15 *  @return None
16 *****************************************************/
17 void storage_init(storage_t *storage);
18
19
20 /*****************************************************
21 *  @brief  Writes values into EEPROM
22 *  @param  addr Address to where data are written
23 *  @param  val Data that are going to be written
24
25 *  @return None
26 *****************************************************/
27 void EEPROM_write(uint16_t addr, uint8_t val);
28
29
30 /*****************************************************
31 *  @brief  Reads values from EEPROM
32 *  @param  addr Address from where dara are read
33 *  @return Returns value from EEPROM
34 *****************************************************/
35 uint8_t EEPROM_read(uint16_t addr);
36
37
38 /*****************************************************
39 *  @brief  Writes data into EEPROM via
40 *          EEPROM_write function
41 *  @param  data Actual measured data
42 *  @param  storage Storage setup
43 *  @return None
44 *****************************************************/
45 void storage_write(storage_t *storage, dataset_t *data);
46
47
48 /*****************************************************
49 *  @brief  Reads data into EEPROM via
50 *          EEPROM_read function
51 *  @param  data Saved data
52 *  @param  storage Storage setup
53 *  @param  pos Position in EEPROM that says
54 *          from where to start read
55 *  @return None
56 *****************************************************/
57 void storage_read(storage_t *storage, dataset_t *data, uint8_t pos);
58
59 //size_t storage_read_multiple(storage_t *storage, dataset_t **data, uint32_t n); // needed?
60
61
62 #endif
63
```

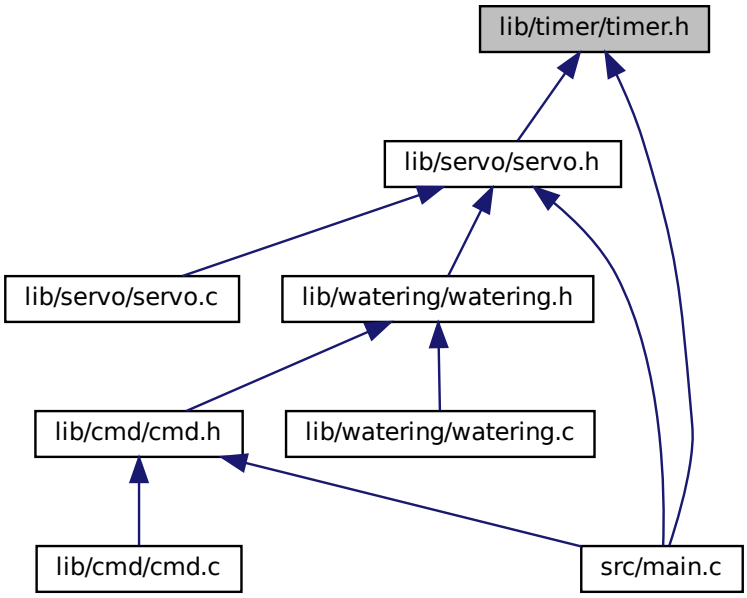
6.29 lib/timer/timer.h File Reference

```
#include <avr/io.h>
```

Include dependency graph for timer.h:



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



Macros

- #define TCCRXB_MODIFY_CS(reg, val) reg = (reg & 0b111) | val

Definitions for 16-bit Timer/Counter1

Note

t_OVF = 1/F_CPU ∗ prescaler ∗ 2^n where n = 16, F_CPU = 16 MHz

- #define TIM0_STOP() TCCRXB_MODIFY_CS(TCCR0B, 0)
  Stop timer, prescaler 000 --> STOP.
- #define TIM0_OVF_16US() TCCRXB_MODIFY_CS(TCCR0B, 1)
  Set overflow 16us, prescaler 001 --> 1.
- #define TIM0_OVF_128US() TCCRXB_MODIFY_CS(TCCR0B, 2)
  Set overflow ms, prescaler 010 --> 8.
- #define TIM0_OVF_1MS() TCCRXB_MODIFY_CS(TCCR0B, 3)
  Set overflow 1ms, prescaler 011 --> 64.
- #define TIM0_OVF_4MS() TCCRXB_MODIFY_CS(TCCR0B, 3)
  Set overflow 4ms, 1024 prescaler 100 --> 256.
- #define TIM0_OVF_16MS() TCCRXB_MODIFY_CS(TCCR0B, 5)
  Set overflow 16ms, prescaler // 101 --> 1024.
- #define TIM1_STOP() TCCRXB_MODIFY_CS(TCCR1B, 0)
  Stop timer, prescaler 000 --> STOP.
- #define TIM1_OVF_4MS() TCCRXB_MODIFY_CS(TCCR1B, 1)
  Set overflow 4ms, prescaler 001 --> 1.
- #define TIM1_OVF_33MS() TCCRXB_MODIFY_CS(TCCR1B, 2)
  Set overflow 33ms, prescaler 010 --> 8.
- #define TIM1_OVF_262MS() TCCRXB_MODIFY_CS(TCCR1B, 3)
  Set overflow 262ms, prescaler 011 --> 64.
- #define TIM1_OVF_1SEC() TCCRXB_MODIFY_CS(TCCR1B, 4)
  Set overflow 1s, prescaler 100 --> 256.
- #define TIM1_OVF_4SEC() TCCRXB_MODIFY_CS(TCCR1B, 5)
  Set overflow 4s, prescaler // 101 --> 1024.
- #define TIM2_STOP() TCCRXB_MODIFY_CS(TCCR2B, 0)
  Stop timer, prescaler 000 --> STOP.
- #define TIM2_OVF_16US() TCCRXB_MODIFY_CS(TCCR2B, 1)
  Set overflow 16us, prescaler 001 --> 1.
- #define TIM2_OVF_128US() TCCRXB_MODIFY_CS(TCCR2B, 2)
  Set overflow 128us, prescaler 010 --> 8.
- #define TIM2_OVF_512US() TCCRXB_MODIFY_CS(TCCR2B, 3)
  Set overflow 512, prescaler 011 --> 32.
- #define TIM2_OVF_1MS() TCCRXB_MODIFY_CS(TCCR2B, 4)
  Set overflow 1ms, 1024 prescaler 100 --> 64.
- #define TIM2_OVF_2MS() TCCRXB_MODIFY_CS(TCCR2B, 5)
  Set overflow 2ms, prescaler // 101 --> 128.
- #define TIM2_OVF_4MS() TCCRXB_MODIFY_CS(TCCR2B, 6)
  Set overflow 4ms, prescaler // 110 --> 256.
- #define TIM2_OVF_16MS() TCCRXB_MODIFY_CS(TCCR2B, 7)
  Set overflow 16ms, prescaler // 111 --> 1024.
- #define TIM0_OVF_ENABLE() TIMSK0 |= (1<<TOIE0)
  Enable overflow interrupt, 1 --> enable.
- #define TIM0_OVF_DISABLE() TIMSK0 &= ~(1<<TOIE0)
  Disable overflow interrupt, 0 --> disable.
- #define TIM1_OVF_ENABLE() TIMSK1 |= (1<<TOIE1)
  Enable overflow interrupt, 1 --> enable.
- #define TIM1_OVF_DISABLE() TIMSK1 &= ~(1<<TOIE1)
  Disable overflow interrupt, 0 --> disable.
- #define TIM2_OVF_ENABLE() TIMSK2 |= (1<<TOIE2)
  Enable overflow interrupt, 1 --> enable.
- #define TIM2_OVF_DISABLE() TIMSK2 &= ~(1<<TOIE2)
  Disable overflow interrupt, 0 --> disable.

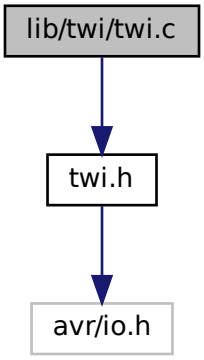
6.30 timer.h

Go to the documentation of this file.

```
1 #ifndef TIMER_H
2 # define TIMER_H
3
4 /*****************************************************************
5 *
6 * Timer library for AVR-GCC.
7 *
8 * ATmega328P (Arduino Uno), 16 MHz, PlatformIO
9 *
10 * Copyright (c) 2019 Tomas Fryza
11 * Dept.  of Radio Electronics, Brno University of Technology, Czechia
12 * This work is licensed under the terms of the MIT license.
13 *
14 *****************************************************************/
15
35 /* Includes ------------------------------------------------------------*/
36 #include <avr/io.h>
37
38 #define TCCRXB_MODIFY_CS(reg, val) reg = (reg & 0b111) | val
39
40 /* Defines -------------------------------------------------------------*/
46 #define TIM0_STOP()       TCCRXB_MODIFY_CS(TCCR0B, 0)
48 #define TIM0_OVF_16US()   TCCRXB_MODIFY_CS(TCCR0B, 1)
50 #define TIM0_OVF_128US()  TCCRXB_MODIFY_CS(TCCR0B, 2)
52 #define TIM0_OVF_1MS()    TCCRXB_MODIFY_CS(TCCR0B, 3)
54 #define TIM0_OVF_4MS()    TCCRXB_MODIFY_CS(TCCR0B, 3)
56 #define TIM0_OVF_16MS()   TCCRXB_MODIFY_CS(TCCR0B, 5)
57
58
60 #define TIM1_STOP()      TCCRXB_MODIFY_CS(TCCR1B, 0)
62 #define TIM1_OVF_4MS()   TCCRXB_MODIFY_CS(TCCR1B, 1)
64 #define TIM1_OVF_33MS()  TCCRXB_MODIFY_CS(TCCR1B, 2)
66 #define TIM1_OVF_262MS() TCCRXB_MODIFY_CS(TCCR1B, 3)
68 #define TIM1_OVF_1SEC()  TCCRXB_MODIFY_CS(TCCR1B, 4)
70 #define TIM1_OVF_4SEC()  TCCRXB_MODIFY_CS(TCCR1B, 5)
71
72
74 #define TIM2_STOP()       TCCRXB_MODIFY_CS(TCCR2B, 0)
76 #define TIM2_OVF_16US()   TCCRXB_MODIFY_CS(TCCR2B, 1)
78 #define TIM2_OVF_128US()  TCCRXB_MODIFY_CS(TCCR2B, 2)
80 #define TIM2_OVF_512US()  TCCRXB_MODIFY_CS(TCCR2B, 3)
82 #define TIM2_OVF_1MS()    TCCRXB_MODIFY_CS(TCCR2B, 4)
84 #define TIM2_OVF_2MS()    TCCRXB_MODIFY_CS(TCCR2B, 5)
86 #define TIM2_OVF_4MS()    TCCRXB_MODIFY_CS(TCCR2B, 6)
88 #define TIM2_OVF_16MS()   TCCRXB_MODIFY_CS(TCCR2B, 7)
89
90
92 #define TIM0_OVF_ENABLE()  TIMSK0 |= (1<<TOIE0)
94 #define TIM0_OVF_DISABLE() TIMSK0 &= ~(1<<TOIE0)
95
97 #define TIM1_OVF_ENABLE()  TIMSK1 |= (1<<TOIE1)
99 #define TIM1_OVF_DISABLE() TIMSK1 &= ~(1<<TOIE1)
100
102 #define TIM2_OVF_ENABLE()  TIMSK2 |= (1<<TOIE2)
104 #define TIM2_OVF_DISABLE() TIMSK2 &= ~(1<<TOIE2)
105
106
111 // WRITE YOUR CODE HERE
112
113
118 // WRITE YOUR CODE HERE
119
120
123 #endif
```

6.31 lib/twi/twi.c File Reference

```
#include <twi.h>
```
Include dependency graph for twi.c:



Macros

- #define TWI_TIMEOUT 100000

Functions

- void twi_init (void)

  Initialize TWI unit, enable internal pull-ups, and set SCL frequency.
- void twi_start (void)

  Start communication on I2C/TWI bus.
- uint8_t twi_write (uint8_t data)

  Send one byte to I2C/TWI Slave device.
- uint8_t twi_read (uint8_t ack)

  Read one byte from I2C/TWI Slave device and acknowledge it by ACK or NACK.
- void twi_stop (void)

  Generates Stop condition on I2C/TWI bus.
- uint8_t twi_test_address (uint8_t adr)

  Test presence of one I2C device on the bus.

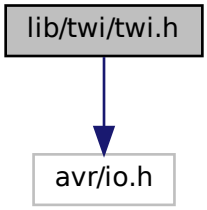
6.31.1 Macro Definition Documentation

6.31.1.1 TWI_TIMEOUT

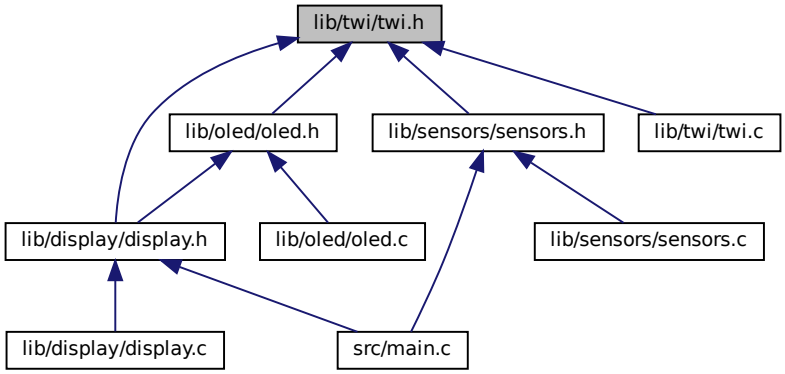
```
#define TWI_TIMEOUT 100000
```

6.32 lib/twi/twi.h File Reference

`#include <avr/io.h>`
Include dependency graph for twi.h:



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



Macros

Definition of frequencies

- #define F_CPU 16000000
  CPU frequency in Hz required TWI_BIT_RATE_REG.
- #define F_SCL 100000
  I2C/TWI bit rate. Must be greater than 31000.
- #define TWI_BIT_RATE_REG ((F_CPU/F_SCL - 16) / 2)
  TWI bit rate register value.

Definition of ports and pins

- #define TWI_PORT PORTC
  Port of TWI unit.
- #define TWI_SDA_PIN 4
  SDA pin of TWI unit.
- #define TWI_SCL_PIN 5
  SCL pin of TWI unit.

Other definitions

- #define TWI_WRITE 0

  Mode for writing to I2C/TWI device.
- #define TWI_READ 1

  Mode for reading from I2C/TWI device.
- #define TWI_ACK 0

  ACK value for writing to I2C/TWI bus.
- #define TWI_NACK 1

  NACK value for writing to I2C/TWI bus.
- #define DDR(_x) (*(&_x - 1))

  Address of Data Direction Register of port _x.
- #define PIN(_x) (*(&_x - 2))

  Address of input register of port _x.
- void twi_init (void)

  Initialize TWI unit, enable internal pull-ups, and set SCL frequency.
- void twi_start (void)

  Start communication on I2C/TWI bus.
- uint8_t twi_write (uint8_t data)

  Send one byte to I2C/TWI Slave device.
- uint8_t twi_read (uint8_t ack)

  Read one byte from I2C/TWI Slave device and acknowledge it by ACK or NACK.
- void twi_stop (void)

  Generates Stop condition on I2C/TWI bus.
- uint8_t twi_test_address (uint8_t adr)

  Test presence of one I2C device on the bus.

6.33 twi.h

Go to the documentation of this file.

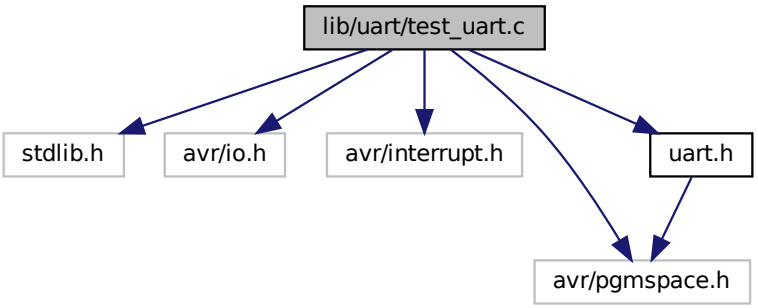
```
1 #ifndef TWI_H
2 # define TWI_H
3
4 /************************************************************************
5 *
6 * I2C/TWI library for AVR-GCC.
7 *
8 * ATmega328P (Arduino Uno), 16 MHz, PlatformIO
9 *
10 * Copyright (c) 2018 Tomas Fryza
11 * Dept.  of Radio Electronics, Brno University of Technology, Czechia
12 * This work is licensed under the terms of the MIT license.
13 *
14 ************************************************************************/
15
35 /* Includes ----------------------------------------------------------*/
36 #include <avr/io.h>
37
38
39 /* Defines -----------------------------------------------------------*/
43 #ifndef F_CPU
44 # define F_CPU 16000000
45 #endif
46 #define F_SCL 100000
47 #define TWI_BIT_RATE_REG ((F_CPU/F_SCL - 16) / 2)
53 #define TWI_PORT PORTC
54 #define TWI_SDA_PIN 4
55 #define TWI_SCL_PIN 5
61 #define TWI_WRITE 0
62 #define TWI_READ 1
63 #define TWI_ACK 0
64 #define TWI_NACK 1
65 #define DDR(_x) (*(&_x - 1))
```

```
66 #define PIN(_x) (*(&_x - 2))
69 /* Function prototypes ------------------------------------------------*/
79 void twi_init(void);
80
81
86 void twi_start(void);
87
88
100 uint8_t twi_write(uint8_t data);
101
102
109 uint8_t twi_read(uint8_t ack);
110
111
116 void twi_stop(void);
117
118
126 uint8_t twi_test_address(uint8_t adr);
127
128
131 #endif
```

6.34 lib/uart/test_uart.c File Reference

```
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include "uart.h"
```

Include dependency graph for test_uart.c:



Macros

- #define UART_BAUD_RATE 9600

Functions

- int main (void)

6.34.1 Macro Definition Documentation

6.34.1.1 UART_BAUD_RATE

```
#define UART_BAUD_RATE 9600
```

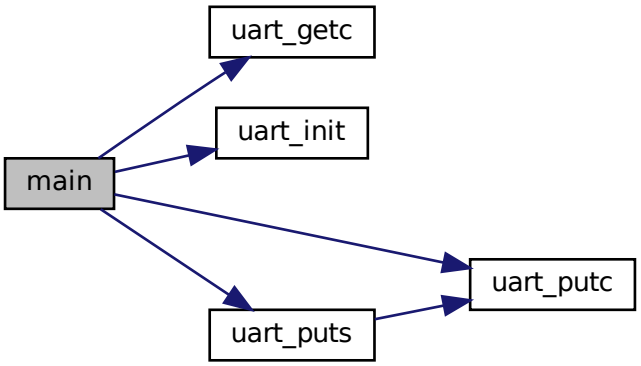
6.34.2 Function Documentation

6.34.2.1 main()

```
int main (
              void  )
30 {
31     unsigned int c;
32     char buffer[7];
33     int  num=134;
34
35
36     /*
37 *  Initialize UART library, pass baudrate and AVR cpu clock
38 *  with the macro
39 *  UART_BAUD_SELECT() (normal speed mode )
40 *  or
41 *  UART_BAUD_SELECT_DOUBLE_SPEED() ( double speed mode)
42 */
43     uart_init( UART_BAUD_SELECT(UART_BAUD_RATE,F_CPU) );
44
45     /*
46 * now enable interrupt, since UART library is interrupt controlled
47 */
48     sei();
49
50     /*
51 *  Transmit string to UART
52 *  The string is buffered by the uart library in a circular buffer
53 *  and one character at a time is transmitted to the UART using interrupts.
54 *  uart_puts() blocks if it can not write the whole string to the circular
55 *  buffer
56 */
57     uart_puts("String stored in SRAM\n");
58
59     /*
60 * Transmit string from program memory to UART
61 */
62     uart_puts_P("String stored in FLASH\n");
63
64
65     /*
66 * Use standard avr-libc functions to convert numbers into string
67 * before transmitting via UART
68 */
69     itoa( num, buffer, 10);   // convert interger into string (decimal format)
70     uart_puts(buffer);        // and transmit string to UART
71
72
73     /*
74 * Transmit single character to UART
75 */
76     uart_putc('\r');
77
78     for(;;)
79     {
80         /*
81 * Get received character from ringbuffer
82 * uart_getc() returns in the lower byte the received character and
83 * in the higher byte (bitmask) the last receive error
84 * UART_NO_DATA is returned when no data is available.
85 *
86 */
87         c = uart_getc();
88         if ( c & UART_NO_DATA )
89         {
90             /*
91 * no data available from UART
92 */
```

```
93         }
94         else
95         {
96             /*
97 * new data available from UART
98 * check for Frame or Overrun error
99 */
100             if ( c & UART_FRAME_ERROR )
101             {
102                 /* Framing Error detected, i.e no stop bit detected */
103                 uart_puts_P("UART Frame Error:  ");
104             }
105             if ( c & UART_OVERRUN_ERROR )
106             {
107                 /*
108 * Overrun, a character already present in the UART UDR register was
109 * not read by the interrupt handler before the next character arrived,
110 * one or more received characters have been dropped
111 */
112                 uart_puts_P("UART Overrun Error:  ");
113             }
114             if ( c & UART_BUFFER_OVERFLOW )
115             {
116                 /*
117 * We are not reading the receive buffer fast enough,
118 * one or more received character have been dropped
119 */
120                 uart_puts_P("Buffer overflow error:  ");
121             }
122             /*
123 * send received character back
124 */
125             uart_putc( (unsigned char)c );
126         }
127     }
128
129 }
```

Here is the call graph for this function:

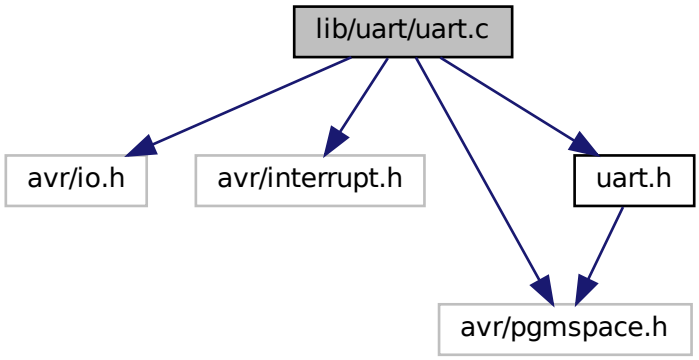


6.35 lib/uart/uart.c File Reference

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
```

```
#include "uart.h"
```

Include dependency graph for uart.c:



Macros

- #define UART_RX_BUFFER_MASK (UART_RX_BUFFER_SIZE - 1)
- #define UART_TX_BUFFER_MASK (UART_TX_BUFFER_SIZE - 1)

Functions

- if (tmphead==UART_RxTail)
- if (UART_TxHead !=UART_TxTail)
- void uart_init (unsigned int baudrate)

  Initialize UART and set baudrate.
- unsigned int uart_getc (void)

  Get received byte from ringbuffer.
- void uart_putc (unsigned char data)

  Put byte to ringbuffer for transmitting via UART.
- void uart_puts (const char ∗s)

  Put string to ringbuffer for transmitting via UART.
- void uart_puts_p (const char ∗progmem_s)

  Put string from program memory to ringbuffer for transmitting via UART.

Variables

- static volatile unsigned char UART_TxBuf [UART_TX_BUFFER_SIZE]
- static volatile unsigned char UART_RxBuf [UART_RX_BUFFER_SIZE] = data
- static volatile unsigned char UART_TxHead
- static volatile unsigned char UART_TxTail
- static volatile unsigned char UART_RxHead
- static volatile unsigned char UART_RxTail
- static volatile unsigned char UART_LastRxError = lastRxError
- unsigned char data
- unsigned char usr = UART0_STATUS
- unsigned char lastRxError
- tmphead = (UART_RxHead + 1) & UART_RX_BUFFER_MASK
- else

6.35.1 Macro Definition Documentation

6.35.1.1 UART_RX_BUFFER_MASK

```
#define UART_RX_BUFFER_MASK ( UART_RX_BUFFER_SIZE - 1)
```

6.35.1.2 UART_TX_BUFFER_MASK

```
#define UART_TX_BUFFER_MASK ( UART_TX_BUFFER_SIZE - 1)
```

6.35.2 Function Documentation

6.35.2.1 if() [1/2]

```
if (
             tmphead  = = UART_RxTail )
392                                  {
393         /* error:  receive buffer overflow */
394         lastRxError = UART_BUFFER_OVERFLOW » 8;
395     }else{
```

6.35.2.2 if() [2/2]

```
if (
             UART_TxHead !     = UART_TxTail )
414                                   {
415         /* calculate and store new buffer index */
416         tmptail = (UART_TxTail + 1) & UART_TX_BUFFER_MASK;
417         UART_TxTail = tmptail;
418         /* get one byte from buffer and write it to UART */
419         UART0_DATA = UART_TxBuf[tmptail];  /* start transmission */
420     }else{
```

6.35.3 Variable Documentation

6.35.3.1 data

```
data
```

Initial value:

```
{
    unsigned char tmphead
```

Referenced by cmd_handler(), display_show_data(), oled_data(), oled_putc(), sensors_update_dataset(), storage_read(), storage_write(), twi_write(), uart_getc(), uart_putc(), and watering_handler().

6.35.3.2 else

```
else
```

Initial value:

```
{
        UART_RxHead = tmphead
```

6.35.3.3 lastRxError

```
unsigned char lastRxError
```

Referenced by if(), and uart_getc().

6.35.3.4 tmphead

```
tmphead = ( UART_RxHead + 1) & UART_RX_BUFFER_MASK
```

Referenced by uart_putc().

6.35.3.5 UART_LastRxError

```
UART_LastRxError = lastRxError  [static]
```

Referenced by uart_getc().

6.35.3.6 UART_RxBuf

```
UART_RxBuf[tmphead] = data  [static]
```

Referenced by uart_getc().

6.35.3.7 UART_RxHead

```
volatile unsigned char UART_RxHead  [static]
```

Referenced by uart_getc(), and uart_init().

6.35.3.8 UART_RxTail

```
volatile unsigned char UART_RxTail  [static]
```

Referenced by uart_getc(), and uart_init().

6.35.3.9 UART_TxBuf

```
volatile unsigned char UART_TxBuf[UART_TX_BUFFER_SIZE]  [static]
```

Referenced by if(), and uart_putc().

6.35.3.10 UART_TxHead

```
volatile unsigned char UART_TxHead  [static]
```

Referenced by uart_init(), and uart_putc().

6.35.3.11 UART_TxTail

```
volatile unsigned char UART_TxTail  [static]
```

Referenced by if(), uart_init(), and uart_putc().

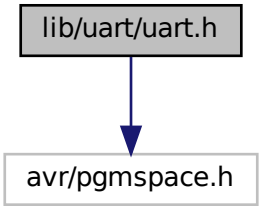
6.35.3.12 usr

```
usr = UART0_STATUS
```

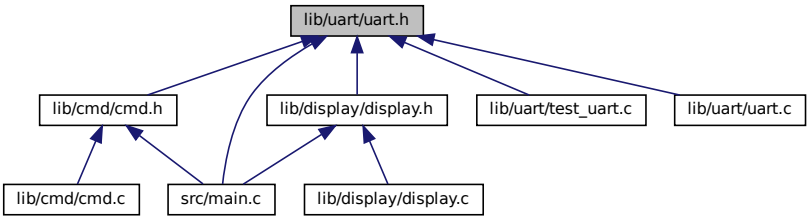
6.36 lib/uart/uart.h File Reference

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

Include dependency graph for uart.h:



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



Macros

- #define UART_BAUD_SELECT(baudRate, xtalCpu) (((xtalCpu) + 8UL ∗ (baudRate)) / (16UL ∗ (baudRate)) -1UL)

  UART Baudrate Expression.
- #define UART_BAUD_SELECT_DOUBLE_SPEED(baudRate, xtalCpu) (((((xtalCpu) + 4UL ∗ (baudRate)) / (8UL ∗ (baudRate)) -1UL)) | 0x8000)

  UART Baudrate Expression for ATmega double speed mode.
- #define UART_RX_BUFFER_SIZE 32

  Size of the circular receive buffer, must be power of 2.
- #define UART_TX_BUFFER_SIZE 256

  Size of the circular transmit buffer, must be power of 2.
- #define UART_FRAME_ERROR 0x1000

Framing Error by UART

- #define UART_OVERRUN_ERROR 0x0800

  Overrun condition by UART

- #define UART_PARITY_ERROR 0x0400

  Parity Error by UART

- #define UART_BUFFER_OVERFLOW 0x0200

  receive ringbuffer overflow

- #define UART_NO_DATA 0x0100

  no receive data available

- #define uart_puts_P(__s) uart_puts_p(PSTR(__s))

  Macro to automatically put a string constant into program memory.

- #define uart1_puts_P(__s) uart1_puts_p(PSTR(__s))

  Macro to automatically put a string constant into program memory.

Functions

- void uart_init (unsigned int baudrate)

  Initialize UART and set baudrate.

- unsigned int uart_getc (void)

  Get received byte from ringbuffer.

- void uart_putc (unsigned char data)

  Put byte to ringbuffer for transmitting via UART.

- void uart_puts (const char *s)

  Put string to ringbuffer for transmitting via UART.

- void uart_puts_p (const char *s)

  Put string from program memory to ringbuffer for transmitting via UART.

- void uart1_init (unsigned int baudrate)

  Initialize USART1 (only available on selected ATmegas)

- unsigned int uart1_getc (void)

  Get received byte of USART1 from ringbuffer. (only available on selected ATmega)

- void uart1_putc (unsigned char data)

  Put byte to ringbuffer for transmitting via USART1 (only available on selected ATmega)

- void uart1_puts (const char *s)

  Put string to ringbuffer for transmitting via USART1 (only available on selected ATmega)

- void uart1_puts_p (const char *s)

  Put string from program memory to ringbuffer for transmitting via USART1 (only available on selected ATmega)

6.37 uart.h

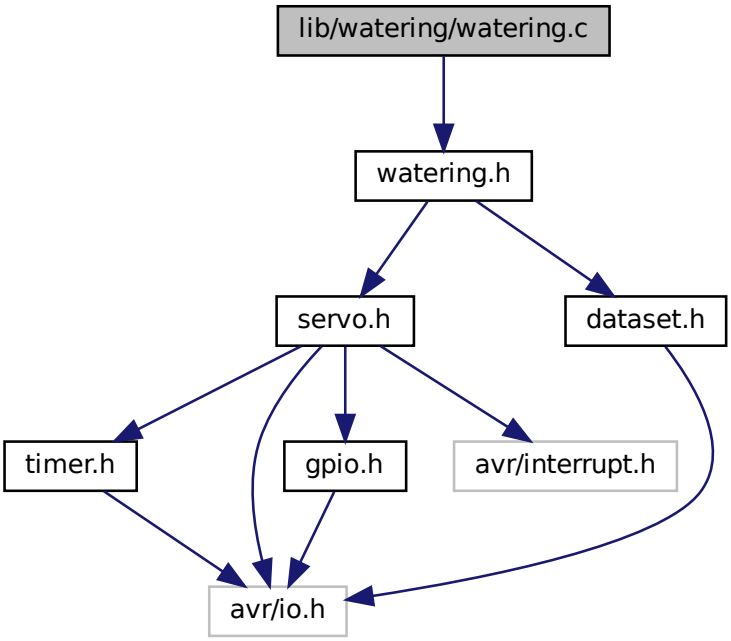
Go to the documentation of this file.

```
1 #ifndef UART_H
2 #define UART_H
3 /************************************************************************
4 Title:        Interrupt UART library with receive/transmit circular buffers
5 Author:      Peter Fleury <pfleury@gmx.ch>  http://tinyurl.com/peterfleury
6 File:          $Id:  uart.h,v 1.13 2015/01/11 13:53:25 peter Exp $
7 Software:  AVR-GCC 4.x, AVR Libc 1.4 or higher
8 Hardware:  any AVR with built-in UART/USART
9 Usage:       see Doxygen manual
```

```
10
11 LICENSE:
12 Copyright (C) 2015 Peter Fleury, GNU General Public License Version 3
13
14 This program is free software; you can redistribute it and/or modify
15 it under the terms of the GNU General Public License as published by
16 the Free Software Foundation; either version 3 of the License, or
17 any later version.
18
19 This program is distributed in the hope that it will be useful,
20 but WITHOUT ANY WARRANTY; without even the implied warranty of
21 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
22 GNU General Public License for more details.
23
24 ************************************************************************/
25
50 #include <avr/pgmspace.h>
51
52 #if (__GNUC__ * 100 + __GNUC_MINOR__) < 405
53 #error "This library requires AVR-GCC 4.5 or later, update to newer AVR-GCC compiler !"
54 #endif
55
56
60 /*
61 ** constants and macros
62 */
63
64
69 #define UART_BAUD_SELECT(baudRate,xtalCpu)  (((xtalCpu) + 8UL * (baudRate)) / (16UL * (baudRate)) -1UL)
70
75 #define UART_BAUD_SELECT_DOUBLE_SPEED(baudRate,xtalCpu) ( ((((xtalCpu) + 4UL * (baudRate)) / (8UL *
       (baudRate)) -1UL)) | 0x8000)
76
82 #ifndef UART_RX_BUFFER_SIZE
83 #define UART_RX_BUFFER_SIZE 32
84 #endif
85
91 #ifndef UART_TX_BUFFER_SIZE
92 #define UART_TX_BUFFER_SIZE 256
93 #endif
94
95 /* test if the size of the circular buffers fits into SRAM */
96 #if ( (UART_RX_BUFFER_SIZE+UART_TX_BUFFER_SIZE) >= (RAMEND-0x60 ) )
97 #error "size of UART_RX_BUFFER_SIZE + UART_TX_BUFFER_SIZE larger than size of SRAM"
98 #endif
99
100 /*
101 ** high byte error return code of uart_getc()
102 */
103 #define UART_FRAME_ERROR      0x1000
104 #define UART_OVERRUN_ERROR    0x0800
105 #define UART_PARITY_ERROR     0x0400
106 #define UART_BUFFER_OVERFLOW  0x0200
107 #define UART_NO_DATA          0x0100
110 /*
111 ** function prototypes
112 */
113
119 extern void uart_init(unsigned int baudrate);
120
121
146 extern unsigned int uart_getc(void);
147
148
154 extern void uart_putc(unsigned char data);
155
156
167 extern void uart_puts(const char *s );
168
169
181 extern void uart_puts_p(const char *s );
182
186 #define uart_puts_P(__s)       uart_puts_p(PSTR(__s))
187
188
189
191 extern void uart1_init(unsigned int baudrate);
193 extern unsigned int uart1_getc(void);
195 extern void uart1_putc(unsigned char data);
197 extern void uart1_puts(const char *s );
199 extern void uart1_puts_p(const char *s );
201 #define uart1_puts_P(__s)       uart1_puts_p(PSTR(__s))
202
206 #endif // UART_H
207
```

6.38 lib/watering/watering.c File Reference

`#include "watering.h"`
Include dependency graph for watering.c:



Functions

- void watering_init (watering_t *watering, servo_t *servo)
- void watering_set_limit (watering_t *watering, uint16_t min, uint16_t max)
- void watering_handler (watering_t *watering, dataset_t *data)

6.38.1 Function Documentation

6.38.1.1 watering_handler()

```
void watering_handler (
            watering_t * watering,
            dataset_t * data )
16 {
17     if(watering->servo->value != 160 && watering->servo->value != 20)
18         servo_set_value(watering->servo, 20);
19
20     if(watering->servo->value == 20 && data->moist < watering->min)
21         servo_set_value(watering->servo, 160);
```

```
22     if(watering->servo->value != 20 && data->moist > watering->max)
23         servo_set_value(watering->servo, 20);
24 }
```

Referenced by main().

Here is the call graph for this function:

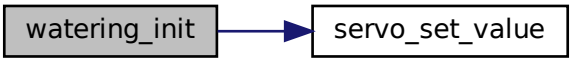
Here is the caller graph for this function:

6.38.1.2 watering_init()

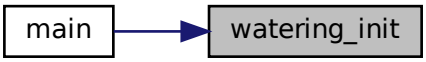
```
void watering_init (
             watering_t * watering,
             servo_t * servo )
4 {
5     watering->servo = servo;
6     servo_set_value(watering->servo, 0);
7 }
```

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



6.38.1.3 watering_set_limit()

```
void watering_set_limit (
              watering_t * watering,
              uint16_t min,
              uint16_t max )
10 {
11     watering->min = min;
12     watering->max = max;
13 }
```

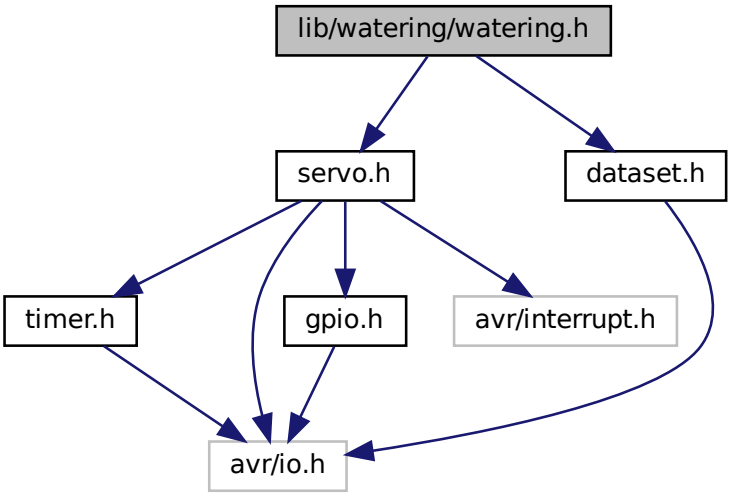
Referenced by main().

Here is the caller graph for this function:

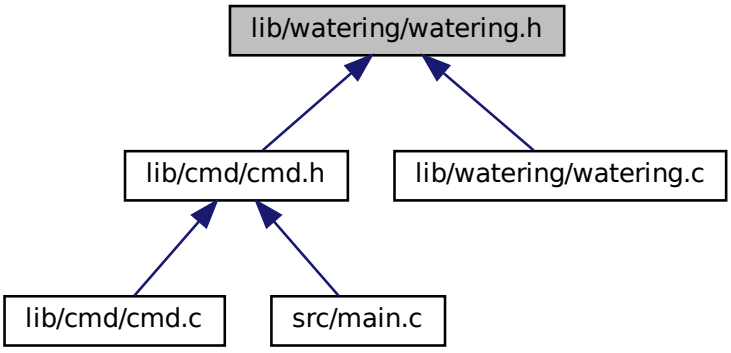
6.39 lib/watering/watering.h File Reference

```
#include <servo.h>
#include <dataset.h>
```

Include dependency graph for watering.h:



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



Classes

- struct watering_t

Functions

- void watering_init (watering_t *watering, servo_t *servo)
- void watering_set_limit (watering_t *watering, uint16_t min, uint16_t max)
- void watering_handler (watering_t *watering, dataset_t *data)

6.39.1 Function Documentation

6.39.1.1 watering_handler()

```
void watering_handler (
            watering_t * watering,
            dataset_t * data )
16 {
17     if(watering->servo->value != 160 && watering->servo->value != 20)
18         servo_set_value(watering->servo, 20);
19
20     if(watering->servo->value == 20 && data->moist < watering->min)
21         servo_set_value(watering->servo, 160);
22     if(watering->servo->value != 20 && data->moist > watering->max)
23         servo_set_value(watering->servo, 20);
24 }
```

Referenced by main().

Here is the call graph for this function:

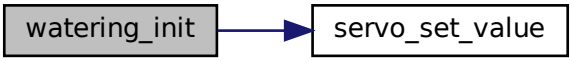
Here is the caller graph for this function:

6.39.1.2 watering_init()

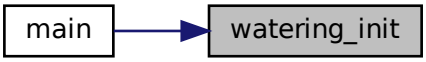
```
void watering_init (
            watering_t * watering,
            servo_t * servo )
4 {
5     watering->servo = servo;
6     servo_set_value(watering->servo, 0);
7 }
```

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



6.39.1.3 watering_set_limit()

```
void watering_set_limit (
             watering_t * watering,
             uint16_t min,
             uint16_t max )
10 {
11     watering->min = min;
12     watering->max = max;
13 }
```

Referenced by main().

Here is the caller graph for this function:

6.40 watering.h

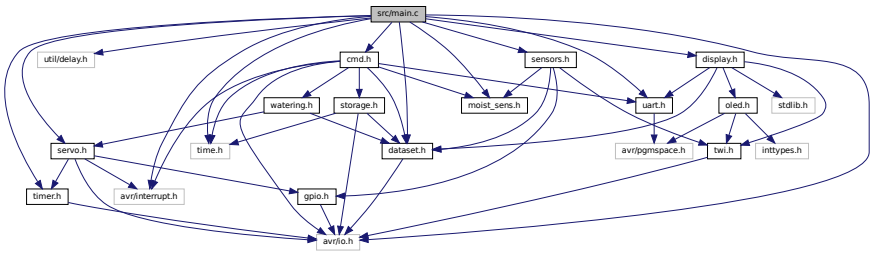
Go to the documentation of this file.

```
1 #ifndef WATERING_H
2 #define WATERING_H
3
4 #include <servo.h>
5 #include <dataset.h>
6
17 typedef struct
18 {
19     servo_t *servo;
20     uint16_t min;
21     uint16_t max;
22 } watering_t;
23
24 /************************************************
25 *  @brief  ---
26 *  @param  watering ---
27 *  @param  servo ---
28 *  @return None
29 ************************************************/
30 void watering_init(watering_t *watering, servo_t *servo);
31
32 /************************************************
33 *  @brief  ---
34 *  @param  watering ---
35 *  @param  min ---
36 *  @param  max ---
37 *  @return None
38 ************************************************/
39 void watering_set_limit(watering_t *watering, uint16_t min, uint16_t max);
40
41 /************************************************
42 *  @brief  ---
43 *  @param  watering ---
44 *  @param  data --- Current measurements
45 *  @return None
46 ************************************************/
47 void watering_handler(watering_t *watering, dataset_t *data);
48
49 #endif
50
```

6.41 src/main.c File Reference

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "timer.h"
#include <time.h>
#include <servo.h>
#include <uart.h>
#include <dataset.h>
#include <cmd.h>
#include <moist_sens.h>
#include <sensors.h>
#include <display.h>
```

Include dependency graph for main.c:



Macros

- #define F_CPU 16000000

Functions

- int main (void)
- ISR (TIMER0_OVF_vect)
- ISR (TIMER1_OVF_vect)

Variables

- time_t last_measurement_time
- servo_t water_servo
- dataset_t actual_data
- watering_t watering
- storage_t storage

6.41.1 Macro Definition Documentation

6.41.1.1 F_CPU

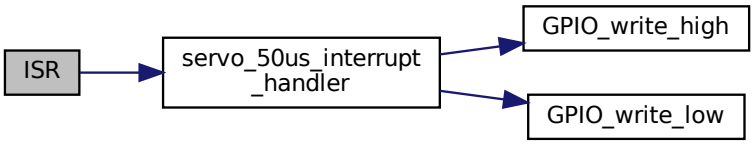
```
#define F_CPU 16000000
```

6.41.2 Function Documentation

6.41.2.1 ISR() [1/2]

```
ISR (
              TIMER0_OVF_vect  )
73 {
74     TCNT0 = 158;
75     servo_50us_interrupt_handler(&water_servo);
76 }
```

Here is the call graph for this function:



6.41.2.2 ISR() [2/2]

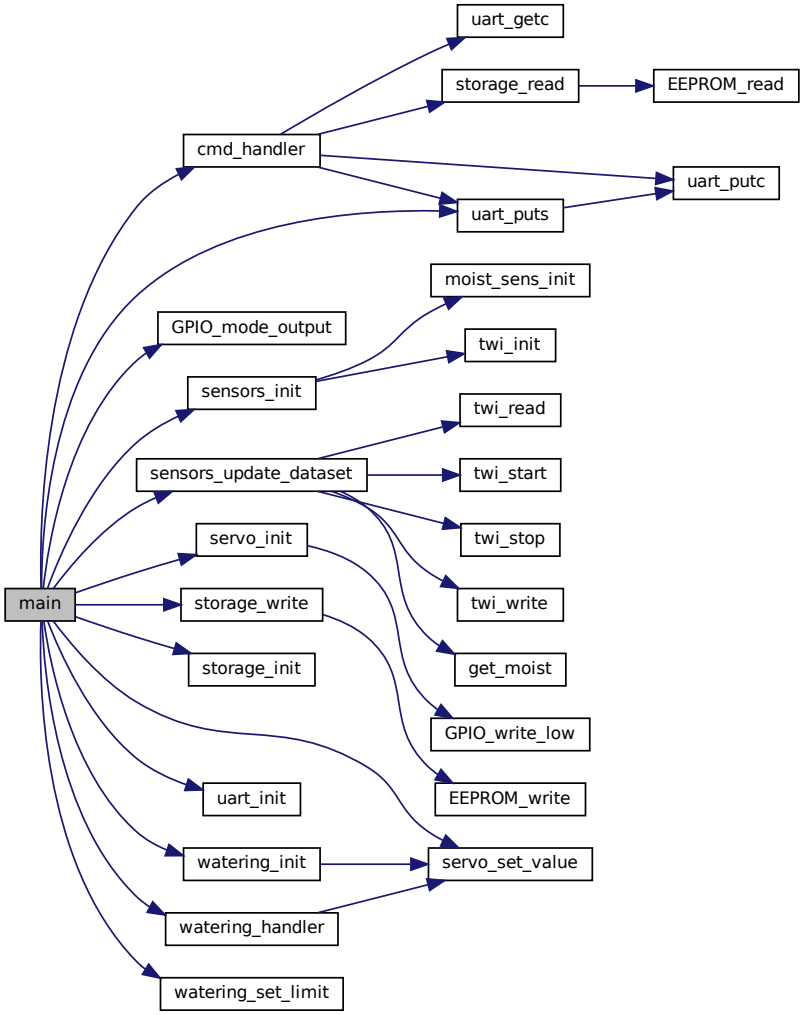
```
ISR (
              TIMER1_OVF_vect  )
79 {
80     actual_data.time++;
81 }
```

6.41.2.3 main()

```
int main (
              void  )
29 {
30     sei();
31     uart_init(UART_BAUD_SELECT(9600, F_CPU));
32
33     //display_init();
34
35     GPIO_mode_output(&DDRB, 0);
36     servo_init(&water_servo, &PORTB, 0);
37     servo_set_value(&water_servo, 0);
38
39     storage_init(&storage);
40     watering_init(&watering, &water_servo);
41     watering_set_limit(&watering, 0, 255);
42     sensors_init();
43
44     TIM1_OVF_1SEC();
45     TIM1_OVF_ENABLE();
46
47     //set_zone(+1*ONE_HOUR);
48     actual_data.time = 0; //1701344319 - AVRTIME_TO_UNIXTIME;
49
50     uart_puts("Watering system terminal (pres ?  for help):");
51
52     while(1)
53     {
```

```
54          cmd_handler(&actual_data, &watering, &storage);
55          _delay_ms(1000);
56          watering_handler(&watering, &actual_data);
57          sensors_update_dataset(&actual_data);
58          //servo_set_value(&water_servo, actual_data.time%90);
59
60          // Every save and display measurement
61          if(last_measurement_time + 10 < actual_data.time)
62          {
63              last_measurement_time = actual_data.time;
64              //display_show_data(&actual_data);
65              storage_write(&storage, &actual_data);
66          }
67      }
68      return 0;
69  }
```

Here is the call graph for this function:



6.41.3 Variable Documentation

6.41.3.1 actual_data

```
dataset_t actual_data
```

Referenced by ISR(), and main().

6.41.3.2 last_measurement_time

```
time_t last_measurement_time
```

Referenced by main().

6.41.3.3 storage

```
storage_t storage
```

Referenced by cmd_handler(), main(), storage_init(), storage_read(), and storage_write().

6.41.3.4 water_servo

```
servo_t water_servo
```

Referenced by ISR(), and main().

6.41.3.5 watering

```
watering_t watering
```

Referenced by cmd_handler(), main(), watering_handler(), watering_init(), and watering_set_limit().