Arduino

1.0

Generated by Doxygen 1.8.14

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 CpuLoad Class Reference

Class defining CPU load libraries.

`#include <CpuLoad.h>`

Public Member Functions

- CpuLoad ()
  CpuLoad class constructor.
- void ComputeCPULoad ()
  Computes current CPU load.
- uint8_t getCurrrentCPULoad ()
  Get current CPU load value.
- uint8_t getAverageCPULoad ()
  Get average CPU load value.
- uint8_t getMaxCPULoad ()
  Get maximum CPU load value.

Private Attributes

- uint8_t current_load
- uint8_t avg_load
- uint8_t max_load
- uint8_t sample_cnt
- uint8_t sample_mem [NB_OF_SAMPLES]
- uint8_t sample_idx
- uint16_t last_sum_value

3.1.1 Detailed Description

Class defining CPU load libraries.

This class defines tools to compute and monitor CPU load.

Definition at line 19 of file CpuLoad.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 CpuLoad()

```
CpuLoad::CpuLoad ( )
```

CpuLoad class constructor.

This function initializes class CpuLoad

Returns

Nothing

Definition at line 13 of file CpuLoad.cpp.

3.1.3 Member Function Documentation

3.1.3.1 ComputeCPULoad()

```
void CpuLoad::ComputeCPULoad ( )
```

Computes current CPU load.

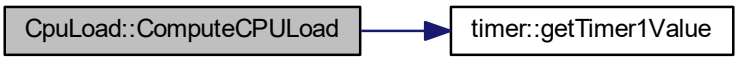
This function computes the current CPU load using value of the timer used by the scheduler at the end of the periodic cycle. This value is divided by the PIT period to obtain CPU load;

Returns

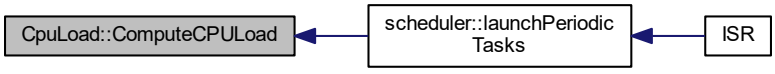
Nothing

Definition at line 27 of file CpuLoad.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.3.2 getAverageCPULoad()

```
uint8_t CpuLoad::getAverageCPULoad ( )  [inline]
```

Get average CPU load value.

This function returns the average CPU load value

Returns

Average CPU load value

Definition at line 56 of file CpuLoad.h.

Here is the caller graph for this function:

3.1.3.3 getCurrrentCPULoad()

```
uint8_t CpuLoad::getCurrrentCPULoad ( )  [inline]
```

Get current CPU load value.

This function returns the current CPU load value

Returns

Current CPU load value

Definition at line 45 of file CpuLoad.h.

Here is the caller graph for this function:

3.1.3.4 getMaxCPULoad()

```
uint8_t CpuLoad::getMaxCPULoad ( )  [inline]
```

Get maximum CPU load value.

This function returns the maximum CPU load value

Returns

Maximum CPU load value

Definition at line 67 of file CpuLoad.h.

Here is the caller graph for this function:

3.1.4 Member Data Documentation

3.1.4.1 avg_load

```
uint8_t CpuLoad::avg_load  [private]
```

Average CPU load based on the last 50 cycles

Definition at line 74 of file CpuLoad.h.

3.1.4.2 current_load

```
uint8_t CpuLoad::current_load  [private]
```

Current CPU load (load of last cycle)

Definition at line 73 of file CpuLoad.h.

3.1.4.3 last_sum_value

```
uint16_t CpuLoad::last_sum_value  [private]
```

Value of the last computed sum (it will reduce the number of samples to sum and speed up execution time)

Definition at line 79 of file CpuLoad.h.

3.1.4.4 max_load

```
uint8_t CpuLoad::max_load  [private]
```

Maximum CPU load since power on

Definition at line 75 of file CpuLoad.h.

3.1.4.5 sample_cnt

```
uint8_t CpuLoad::sample_cnt  [private]
```

Number of samples used to compute average load

Definition at line 76 of file CpuLoad.h.

3.1.4.6 sample_idx

```
uint8_t CpuLoad::sample_idx  [private]
```

Current measurement index (used to memorize the current measure at the correct location in table)

Definition at line 78 of file CpuLoad.h.

3.1.4.7 sample_mem

```
uint8_t CpuLoad::sample_mem[NB_OF_SAMPLES]  [private]
```

Memorization of the last NB_OF_SAMPLES measures

Definition at line 77 of file CpuLoad.h.

The documentation for this class was generated from the following files:

- CpuLoad.h
- CpuLoad.cpp

3.2 dht22 Class Reference

DHT 22 driver class.

`#include <dht22.h>`

Public Member Functions

- dht22 ()
  dht22 class constructor
- bool read (uint16_t *raw_humidity, uint16_t *raw_temperature)
  Reads the data from DHT22.

Private Member Functions

- void initializeCommunication ()
  Initializes the communication.

3.2.1 Detailed Description

DHT 22 driver class.

This class defines all useful functions for DHT22 temperature and humidity sensor

Definition at line 22 of file dht22.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 dht22()

`dht22::dht22 ( )`

dht22 class constructor

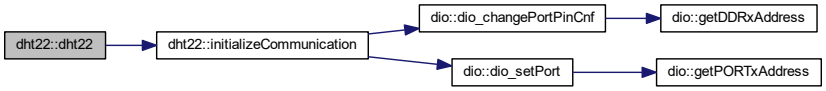
Initializes the class dht22

Returns

Nothing

Definition at line 22 of file dht22.cpp.

Here is the call graph for this function:



3.2.3 Member Function Documentation

3.2.3.1 initializeCommunication()

```
void dht22::initializeCommunication ( )  [private]
```

Initializes the communication.

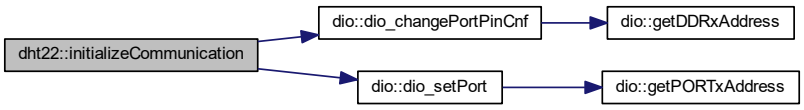
This function initializes the communication with DHT22 using 1-wire protocol

Returns

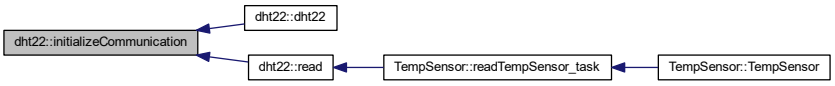
Nothing

Definition at line 198 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.3.2 read()

```
bool dht22::read (
            uint16_t * raw_humidity,
            uint16_t * raw_temperature )
```

Reads the data from DHT22.

This function communicates with DHT22 using 1-wire protocol to read raw values of temperature and humidity. A checksum check is done when communication is finished to validate the received data

Parameters

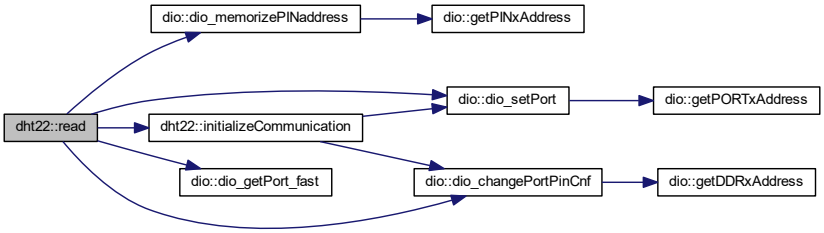
out	*raw_humidity*	Raw humidity value received from sensor
out	*raw_temperature*	Raw temperature value received from sensor

Returns

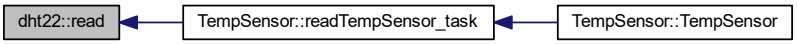
Validity of the read value

Definition at line 27 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following files:

- dht22.h
- dht22.cpp

3.3 dio Class Reference

DIO class.

`#include <dio.h>`

Public Member Functions

- dio ()

  dio class constructor
- void dio_setPort (uint8_t portcode, bool state)

  Port setting function.
- void dio_invertPort (uint8_t portcode)

  Inverts the state of output port.
- bool dio_getPort (uint8_t portcode)

  Gets the logical state of selected pin.
- bool dio_getPort_fast (void)

  Gets the logical state of the memorized pin.
- void dio_changePortPinCnf (uint8_t portcode, uint8_t cnf)

  Changes the IO configuration of the selected pin.
- void dio_memorizePINaddress (uint8_t portcode)

  Memorizes PINx register address and pin index.

Private Member Functions

- void ports_init ()

  Digital ports hardware initialization function.
- uint8_t * getPORTxAddress (uint8_t portcode)

  Gets the physical address of the requested register PORTx.
- uint8_t * getPINxAddress (uint8_t portcode)

  Gets the physical address of the requested register PINx.
- uint8_t * getDDRxAddress (uint8_t portcode)

  Gets the physical address of the requested register DDRx.

Private Attributes

- uint8_t * PINx_addr_mem
- uint8_t PINx_idx_mem

3.3.1 Detailed Description

DIO class.

This class defines all useful functions for digital input/output ports

Definition at line 31 of file dio.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 dio()

```
dio::dio ( )
```

dio class constructor

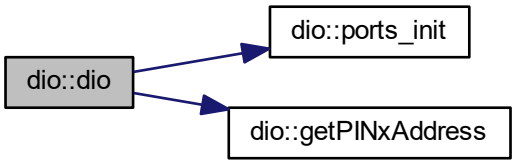
Initializes class dio and calls DIO hardware initialization function

Returns

Nothing

Definition at line 112 of file dio.cpp.

Here is the call graph for this function:



3.3.3 Member Function Documentation

3.3.3.1 dio_changePortPinCnf()

```
void dio::dio_changePortPinCnf (
            uint8_t portcode,
            uint8_t cnf )
```

Changes the IO configuration of the selected pin.

This function configures the selected pin as input or output according to parameter cnf. The corresponding port and pin index is extracted from parameter portcode.

Parameters

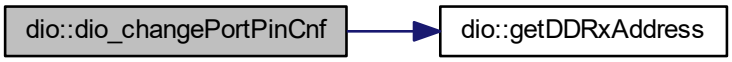
in	*portcode*	Encoded pin and register index
in	*cnf*	Requested configuration for the selected pin PORT_CNF_OUT (1) : pin configured as output PORT_CNF_IN (0) : pin configured as input

Returns

Nothing

Definition at line 149 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.3.3.2 dio_getPort()

```
bool dio::dio_getPort (
            uint8_t portcode )
```

Gets the logical state of selected pin.

This function gets the logical value of the selected pin. The corresponding port and pin index is extracted from parameter portcode.

Parameters

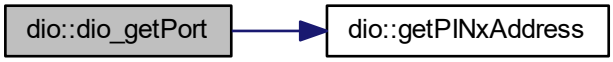
in	*portcode*	Encoded pin and register index

Returns

Logical state of selected pin

Definition at line 139 of file dio.cpp.

Here is the call graph for this function:



3.3.3.3 dio_getPort_fast()

```
bool dio::dio_getPort_fast (
            void  )
```

Gets the logical state of the memorized pin.

This function gets the logical value of the memorized pin. The corresponding port and pin index are stored in class members PINx_addr_mem and PINx_idx_mem. This mechanism is used to speed up reading time as this function no longer needs to extract register address and pin index from portcode.

Returns

Logical state of selected pin

Definition at line 171 of file dio.cpp.

Here is the caller graph for this function:

3.3.3.4 dio_invertPort()

```
void dio::dio_invertPort (
            uint8_t portcode )
```

Inverts the state of output port.

This function inverts the state of the chosen pin. The corresponding port and pin index is extracted from parameter portcode.

Parameters

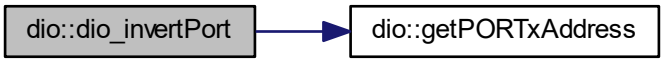
in	*portcode*	Encoded pin and register index

Returns

Nothing

Definition at line 131 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.3.3.5 dio_memorizePINaddress()

```
void dio::dio_memorizePINaddress (
            uint8_t portcode )
```

Memorizes PINx register address and pin index.

This function is used to speed up reading of register PINx. Register address and pin index are decoded from portcode parameter and stored for later use by function dio_getPort_fast.

Parameters

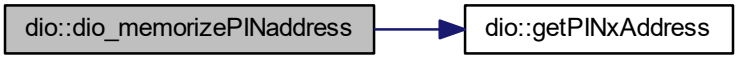
in	*portcode*	Encoded pin and register index

Returns

Nothing

Definition at line 165 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.3.3.6 dio_setPort()

```
void dio::dio_setPort (
            uint8_t portcode,
            bool state )
```

Port setting function.

This function sets the requested digital output to the requested state. The corresponding port and pin index is extracted from parameter portcode.

Parameters

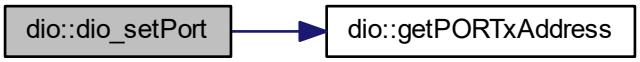
in	*portcode*	Encoded pin and register index
in	*state*	Requested state to set pin

Returns

Nothing

Definition at line 121 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.3.3.7 getDDRxAddress()

```
uint8_t * dio::getDDRxAddress (
            uint8_t portcode )  [private]
```

Gets the physical address of the requested register DDRx.

This function retrieves the address of the register DDRx where x is encoded into the parameter portcode.

Parameters

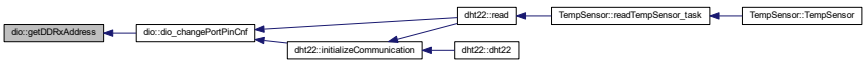
in	*portcode*	Encoded port code

Returns

Pointer to the DDRx register

Definition at line 83 of file dio.cpp.

Here is the caller graph for this function:



3.3.3.8 getPINxAddress()

```
uint8_t * dio::getPINxAddress (
            uint8_t portcode )  [private]
```

Gets the physical address of the requested register PINx.

This function retrieves the address of the register PINx where x is encoded into the parameter portcode.

Parameters

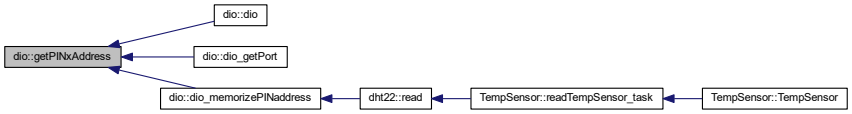
in	*portcode*	Encoded port code

Returns

Pointer to the PINx register

Definition at line 54 of file dio.cpp.

Here is the caller graph for this function:



3.3.3.9 getPORTxAddress()

```
uint8_t * dio::getPORTxAddress (
            uint8_t portcode )  [private]
```

Gets the physical address of the requested register PORTx.

This function retrieves the address of the register PORTx where x is encoded into the parameter portcode.

Parameters

in	*portcode*	Encoded port code

Returns

Pointer to the PORTx register

Definition at line 25 of file dio.cpp.

Here is the caller graph for this function:

3.3.3.10 ports_init()

```
void dio::ports_init ( )  [private]
```

Digital ports hardware initialization function.

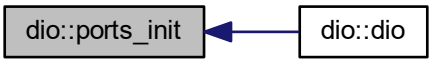
This function initializes digital ports as input or output and sets their initial values

Returns

Nothing

Definition at line 16 of file dio.cpp.

Here is the caller graph for this function:



3.3.4 Member Data Documentation

3.3.4.1 PINx_addr_mem

```
uint8_t* dio::PINx_addr_mem  [private]
```

Memorizes physical address of register PINx in order to speed up register reading time in function dio_getPort_fast

Definition at line 146 of file dio.h.

3.3.4.2 PINx_idx_mem

```
uint8_t dio::PINx_idx_mem  [private]
```

Memorizes pin index of register PINx in order to speed up register reading time in function dio_getPort_fast

Definition at line 147 of file dio.h.

The documentation for this class was generated from the following files:

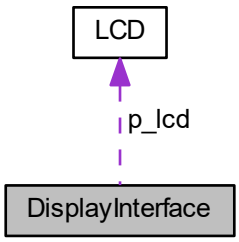
- dio.h
- dio.cpp

3.4 DisplayInterface Class Reference

Display interface services class.

```
#include <DisplayInterface.h>
```

Collaboration diagram for DisplayInterface:



Public Member Functions

- DisplayInterface ()
  Class constructor.
- bool DisplayFullLine (uint8_t *str, uint8_t size, uint8_t line, T_DisplayInterface_LineDisplayMode mode)
  Line display function.
- bool ClearLine (uint8_t line)
  Line clearing function.
- bool IsLineEmpty (uint8_t line)
  Empty line get function.

Private Member Functions

- uint8_t FindFirstCharAddr (uint8_t line)
  Finds start address of a line.

Private Attributes

- LCD * p_lcd
- uint32_t dummy
- bool lineEmptyTab [LCD_SIZE_NB_LINES]

3.4.1 Detailed Description

Display interface services class.

This class defines the services used for interfacing display management services and LCD screen driver

Definition at line 46 of file DisplayInterface.h.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 DisplayInterface()

```
DisplayInterface::DisplayInterface ( )
```

Class constructor.

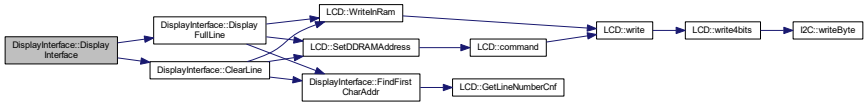
This function initializes all class variables and instantiates the LCD driver according to the defined configuration.

Returns

Nothing

Definition at line 16 of file DisplayInterface.cpp.

Here is the call graph for this function:



3.4.3 Member Function Documentation

3.4.3.1 ClearLine()

```
bool DisplayInterface::ClearLine (
            uint8_t line )
```

Line clearing function.

This function clears the requested line. It sets the corresponding DDRAM addresses to the ASCII value of space character

Parameters

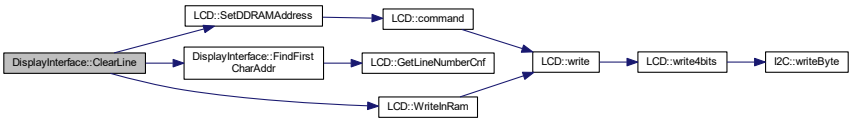
in	*line*	Line to clear

Returns

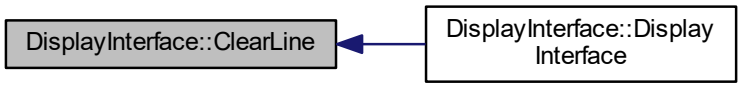
True if the line has been cleared, false otherwise

Definition at line 141 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.2 DisplayFullLine()

```
bool DisplayInterface::DisplayFullLine (
            uint8_t * str,
            uint8_t size,
            uint8_t line,
            T_DisplayInterface_LineDisplayMode mode )
```

Line display function.

This function displays the given string on the requested line. If the string is too long to be displayed entirely, the behavior is defined by the selected mode.

Parameters

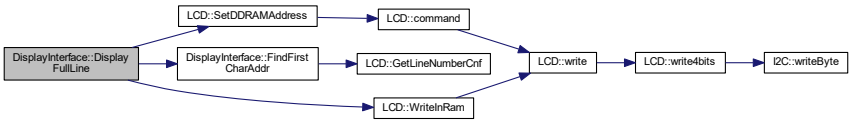
in	*str*	Pointer to the string to display
in	*size*	Size of the string to display
in	*line*	Index of the line where the string shall be displayed
in	*mode*	Display mode

Returns

True if the line has been correctly displayed, false otherwise

Definition at line 44 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.4.3.3 FindFirstCharAddr()

```
uint8_t DisplayInterface::FindFirstCharAddr (
            uint8_t line )  [private]
```

Finds start address of a line.

This function finds the address in DDRAM of the first character of a line.

Parameters

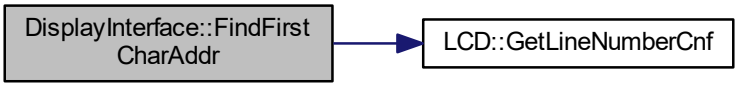
in	*line*	Line which address shall be found

Returns

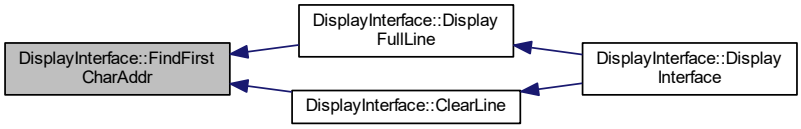
Address in DDRAM of the first character of the line

Definition at line 102 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.4 IsLineEmpty()

```
bool DisplayInterface::IsLineEmpty (
            uint8_t line )
```

Empty line get function.

This function answers if the line given in parameter is empty or not, according to the table isLineEmpty[]

Parameters

in	*line*	Requested line

Returns

True if the line is empty, false otherwise

Definition at line 163 of file DisplayInterface.cpp.

3.4.4 Member Data Documentation

3.4.4.1 dummy

```
uint32_t DisplayInterface::dummy  [private]
```

Needed for data alignment

Definition at line 90 of file DisplayInterface.h.

3.4.4.2 lineEmptyTab

```
bool DisplayInterface::lineEmptyTab[LCD_SIZE_NB_LINES]  [private]
```

Table indicating whether a line is empty or not (true = line empty, false = line not empty

Definition at line 91 of file DisplayInterface.h.

3.4.4.3 p_lcd

```
LCD* DisplayInterface::p_lcd  [private]
```

Pointer to the attached LCD driver object

Definition at line 89 of file DisplayInterface.h.

The documentation for this class was generated from the following files:

- DisplayInterface.h
- DisplayInterface.cpp

3.5 I2C Class Reference

Two-wire serial interface (I2C) class definition.

```
#include <I2C.h>
```

Public Member Functions

- I2C (uint32_t l_bitrate)
  I2C class constructor.
- bool writeByte (uint8_t *data)
  Byte sending function.
- void setTxAddress (uint8_t address)
  Setting function for Tx I2C address.
- void setBitRate (uint32_t l_bitrate)
  Variable bitrate setting function.

Private Member Functions

- void initializeBus ()

  I2C bus initialization.

Private Attributes

- uint8_t tx_address
- uint32_t bitrate

3.5.1 Detailed Description

Two-wire serial interface (I2C) class definition.

This class manages I2C driver.

Definition at line 23 of file I2C.h.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 I2C()

```
I2C::I2C (
            uint32_t l_bitrate )
```

I2C class constructor.

This function initializes the I2C class and calls bus initialization function

Parameters

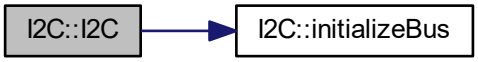
in	*l_bitrate*	Requested bitrate for I2C bus (in Hz)

Returns

Nothing

Definition at line 15 of file I2C.cpp.

Here is the call graph for this function:



3.5.3 Member Function Documentation

3.5.3.1 initializeBus()

```
void I2C::initializeBus ( )  [private]
```

I2C bus initialization.

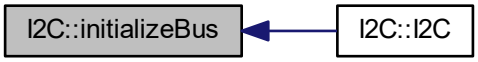
This function initializes the I2C bus, it resets the bus and configure the bitrate as requested. Bitrate is configured according to formula in the ATMEGA2560 datasheet : SCL freq = F_CPU / (16 + 2∗TWBR∗(4^TWPS)). Prescaler value is fixed to 1 (TWPS1 = 0 and TWPS0 = 0), then only TWBR value shall be computed.

Returns

Nothing

Definition at line 76 of file I2C.cpp.

Here is the caller graph for this function:



3.5.3.2 setBitRate()

```
void I2C::setBitRate (
            uint32_t l_bitrate )
```

Variable bitrate setting function.

This function sets the class variable bitrate as requested in parameter.

Parameters

in	*l_bitrate*	Requested bitrate (in Hz)

Returns

Nothing

Definition at line 71 of file I2C.cpp.

3.5.3.3 setTxAddress()

```
void I2C::setTxAddress (
            uint8_t address )
```

Setting function for Tx I2C address.

This function sets the given Tx I2C address in the internal class variable.

Parameters

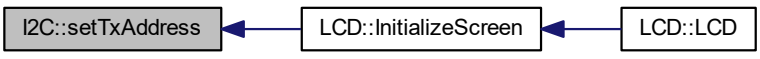
in	*address*	Requested Tx address

Returns

Nothing

Definition at line 66 of file I2C.cpp.

Here is the caller graph for this function:



3.5.3.4 writeByte()

```
bool I2C::writeByte (
            uint8_t * data )
```

Byte sending function.

This function sends one byte on I2C bus

Parameters

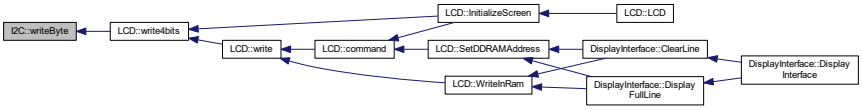
in	*data*	Pointer to the data to send

Returns

True if transmission is completed, False if an error has occurred

Definition at line 23 of file I2C.cpp.

Here is the caller graph for this function:



3.5.4 Member Data Documentation

3.5.4.1 bitrate

```
uint32_t I2C::bitrate  [private]
```

Definition at line 63 of file I2C.h.

3.5.4.2 tx_address

```
uint8_t I2C::tx_address  [private]
```

Definition at line 62 of file I2C.h.

The documentation for this class was generated from the following files:

- I2C.h
- I2C.cpp

3.6 keepAliveLed Class Reference

Class for keep-alive LED blinking.

```
#include <keepAliveLed.h>
```

Public Member Functions

- keepAliveLed ()
  Class constructor.

Static Public Member Functions

- static void blinkLed_task ()
  Task for LED blinking.

3.6.1 Detailed Description

Class for keep-alive LED blinking.

This class defines all functions to make keep-alive LED blink

Definition at line 22 of file keepAliveLed.h.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 keepAliveLed()

```
keepAliveLed::keepAliveLed ( )
```

Class constructor.

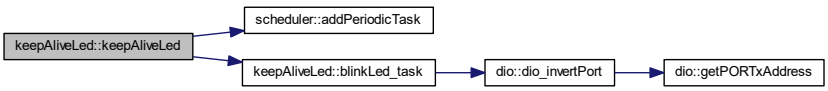
This function initializes the class keppAliveLed

Returns

Nothing

Definition at line 15 of file keepAliveLed.cpp.

Here is the call graph for this function:



3.6.3 Member Function Documentation

3.6.3.1 blinkLed_task()

```
void keepAliveLed::blinkLed_task ( )  [static]
```

Task for LED blinking.

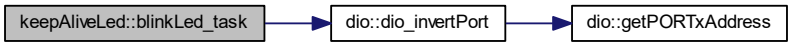
This function is inserted into the scheduler. It changes the state of the LED output to make it blink

Returns

Nothing

Definition at line 21 of file keepAliveLed.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

The documentation for this class was generated from the following files:

- keepAliveLed.h
- keepAliveLed.cpp

3.7 LCD Class Reference

Class for LCD S2004A display driver.

```
#include <LCD.h>
```

Public Member Functions

- LCD (const T_LCD_conf_struct *init_conf)

  LCD class constructor.
- void command (T_LCD_command cmd)

  LCD command management function.
- void ConfigureBacklight (bool enable)

  Backlight configuration function.
- void ConfigureLineNumber (bool param)

  Line type configuration function.
- void ConfigureFontType (bool param)

  Font configuration function.
- void ConfigureDisplayOnOff (bool param)

  Display configuration function.
- void ConfigureCursorOnOff (bool param)

  Cursor configuration function.
- void ConfigureCursorBlink (bool param)

  Cursor blinking configuration function.
- void ConfigureEntryModeDir (bool param)

  Entry mode direction configuration function.
- void ConfigureEntryModeShift (bool param)

  Entry mode shift configuration function.
- void SetDDRAMAddress (uint8_t addr)

  DDRAM address setting function.
- uint8_t GetDDRAMAddress ()

  DDRAM address get function.
- void WriteInRam (uint8_t a_char, T_LCD_ram_area area)

  Screen RAM write function.
- bool GetLineNumberCnf ()

  Number of line get function.

Private Member Functions

- void write4bits (uint8_t data)

  I2C write function for 4-bits mode.
- void write (uint8_t data, T_LCD_config_mode mode)

  I2C write function.
- void InitializeScreen ()

  Screen configuration function.

Private Attributes

- bool backlight_enable
- bool cnfLineNumber
- bool cnfFontType
- bool cnfDisplayOnOff
- bool cnfCursorOnOff
- bool cnfCursorBlink
- bool cnfEntryModeDir
- bool cnfEntryModeShift
- uint8_t ddram_addr

3.7.1 Detailed Description

Class for LCD S2004A display driver.

This class handles functions managing LCD display S2004a on I2C bus

Definition at line 144 of file LCD.h.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 LCD()

```
LCD::LCD (
            const T_LCD_conf_struct * init_conf )
```

LCD class constructor.

This constructor function initializes the class LCD and calls screen configuration function.

Parameters

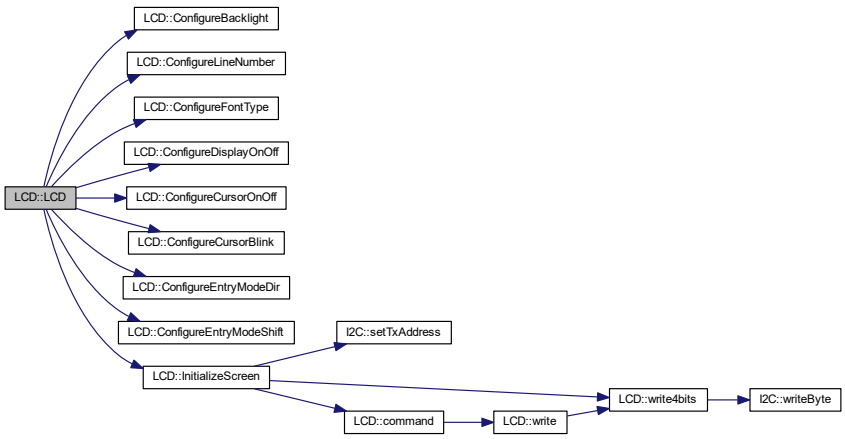
in	*init_conf*	Initial configuration structure

Returns

Nothing

Definition at line 14 of file LCD.cpp.

Here is the call graph for this function:



3.7.3 Member Function Documentation

3.7.3.1 command()

```
void LCD::command (
            T_LCD_command cmd )
```

LCD command management function.

This function sends the requested command to the LCD screen. It builds the 8-bit command word and sends it on I2C bus.

Parameters

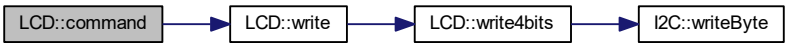
in	*cmd*	Requested command

Returns

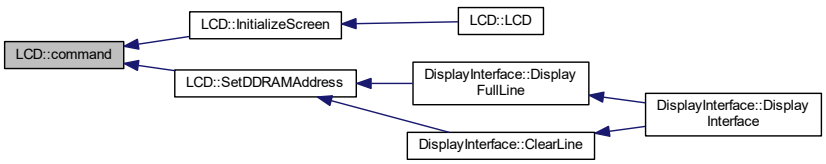
Nothing

Definition at line 114 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.3.2 ConfigureBacklight()

```
void LCD::ConfigureBacklight (
            bool enable )  [inline]
```

Backlight configuration function.

This function configures the screen backlight (enable or disable) according to the parameter enable.

Parameters

in	*enable*	True if backlight shall be on, False otherwise

Returns

Nothing

Definition at line 174 of file LCD.h.

Here is the caller graph for this function:

3.7.3.3 ConfigureCursorBlink()

```
void LCD::ConfigureCursorBlink (
            bool param )  [inline]
```

Cursor blinking configuration function.

This function configures the cursor blinking (on or off mode) according to the parameter.

Parameters

in	*param*	Configuration value

Returns

Nothing

Definition at line 234 of file LCD.h.

Here is the caller graph for this function:

3.7.3.4 ConfigureCursorOnOff()

```
void LCD::ConfigureCursorOnOff (
            bool param )  [inline]
```

Cursor configuration function.

This function configures the cursor (on or off mode) according to the parameter.

Parameters

in	*param*	Configuration value

Returns

Nothing

Definition at line 222 of file LCD.h.

Here is the caller graph for this function:

3.7.3.5 ConfigureDisplayOnOff()

```
void LCD::ConfigureDisplayOnOff (
            bool param )  [inline]
```

Display configuration function.

This function configures the display (on or off mode) according to the parameter.

Parameters

in	*param*	Configuration value

Returns

Nothing

Definition at line 210 of file LCD.h.

Here is the caller graph for this function:

3.7.3.6 ConfigureEntryModeDir()

```
void LCD::ConfigureEntryModeDir (
            bool param )  [inline]
```

Entry mode direction configuration function.

This function configures the direction of entry mode (right or left) according to the parameter.

Parameters

in	*param*	Configuration value

Returns

Nothing

Definition at line 246 of file LCD.h.

Here is the caller graph for this function:

3.7.3.7 ConfigureEntryModeShift()

```
void LCD::ConfigureEntryModeShift (
            bool param )  [inline]
```

Entry mode shift configuration function.

This function configures the display shift of entry mode (enable or disable) according to the parameter.

Parameters

in	*param*	Configuration value

Returns

Nothing

Definition at line 258 of file LCD.h.

Here is the caller graph for this function:

3.7.3.8 ConfigureFontType()

```
void LCD::ConfigureFontType (
            bool param )  [inline]
```

Font configuration function.

This function configures the font type of the screen (5∗8 or 5∗11 dots) according to the parameter.

Parameters

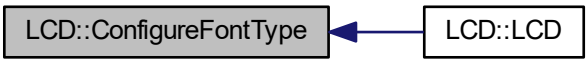
in	*param*	Configuration value

Returns

Nothing

Definition at line 198 of file LCD.h.

Here is the caller graph for this function:



3.7.3.9 ConfigureLineNumber()

```
void LCD::ConfigureLineNumber (
            bool param )  [inline]
```

Line type configuration function.

This function configures the line number configuration of the screen (1 or 2 lines mode) according to the parameter.

Parameters

in	*param*	Configuration value

Returns

Nothing

Definition at line 186 of file LCD.h.

Here is the caller graph for this function:

3.7.3.10 GetDDRAMAddress()

```
uint8_t LCD::GetDDRAMAddress ( )  [inline]
```

DDRAM address get function.

This function return the value of the current DDRAM address stored in internal variable ddram_addr.

Returns

Current DDRAM address

Definition at line 278 of file LCD.h.

3.7.3.11 GetLineNumberCnf()

```
bool LCD::GetLineNumberCnf ( )  [inline]
```

Number of line get function.

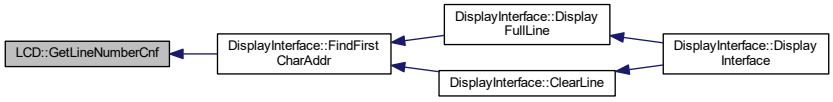
This function returns the line number configuration of the screen : 1 or 2 lines mode.

Returns

Line number configuration

Definition at line 300 of file LCD.h.

Here is the caller graph for this function:



3.7.3.12 InitializeScreen()

```
void LCD::InitializeScreen ( )  [private]
```

Screen configuration function.

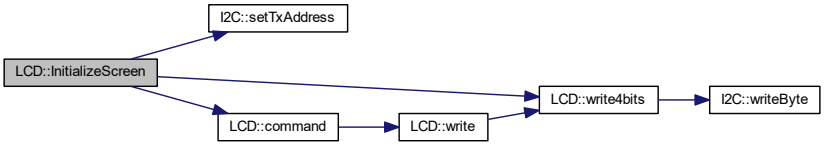
This function configures the LCD screen. It's must be called during initialization phase, or the screen won't be usable. The configuration process is described in LCD datasheet J2004A-GFDN-DYNC

Returns

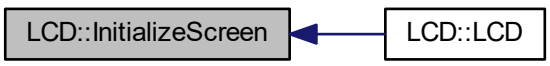
Nothing

Definition at line 62 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.3.13 SetDDRAMAddress()

```
void LCD::SetDDRAMAddress (
            uint8_t addr )
```

DDRAM address setting function.

This function updates the DDRAM address according to the given parameter. The parameter is checked against limits to be sure the address stays always coherent. It also calls the command function to update screen accordingly.

Parameters

in	*addr*	New DDRAM address

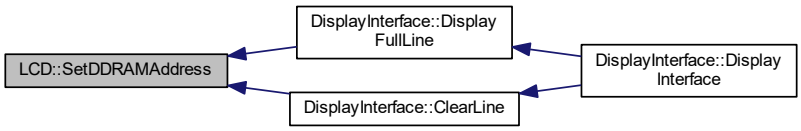
Returns

Nothing

Definition at line 157 of file LCD.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



3.7.3.14 write()

```
void LCD::write (
            uint8_t data,
            T_LCD_config_mode mode )  [private]
```

I2C write function.

This function writes the requested data on I2C bus. It's assumed we only perform write operation so the R/W bit is forced LOW. It's also assumed we work in 4-bit mode, then two calls of write4bits are performed, first with bits 4-7 of data, second with bits 0-3.

Parameters

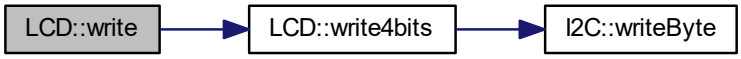
in	*data*	8-bit data for D0-7 pins of screen
in	*mode*	Requested mode for LCD communication

Returns

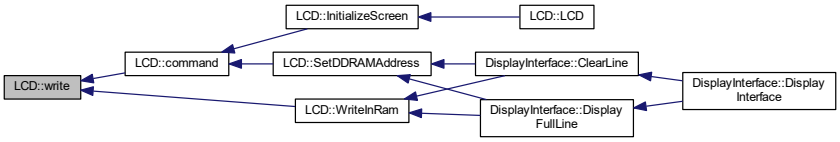
Nothing

Definition at line 51 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.3.15 write4bits()

```
void LCD::write4bits (
            uint8_t data )  [private]
```

I2C write function for 4-bits mode.

This function sends the requested 8-bits data on the I2C bus. The backlight pin is also set/clear according to the configuration. The function sends the data a first time with EN pin set, then a second time with EN bit clear.

Parameters

in	*data*	8-bit data to send

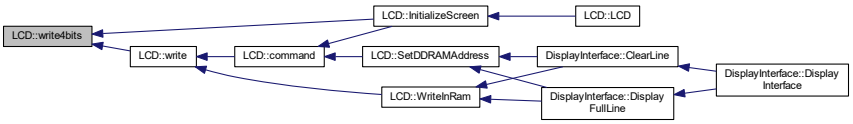
Returns

Nothing

Definition at line 34 of file LCD.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



3.7.3.16 WriteInRam()

```
void LCD::WriteInRam (
            uint8_t a_char,
            T_LCD_ram_area area )
```

Screen RAM write function.

This function writes in the memorized RAM address the character given as parameter. After a write the screen automatically increment/decrement the RAM address, so we do the same in the function to stay coherent. Currently only DDRAM write is implemented.

Parameters

in	*a_char*	Data byte to write in RAM
in	*area*	Area in RAM where the data will be written : DDRAM or CGRAM

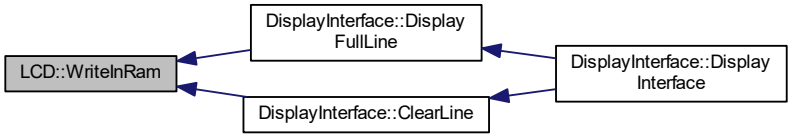
Returns

Nothing

Definition at line 179 of file LCD.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



3.7.4 Member Data Documentation

3.7.4.1 backlight_enable

```
bool LCD::backlight_enable  [private]
```

Backlight enable flag

Definition at line 308 of file LCD.h.

3.7.4.2 cnfCursorBlink

```
bool LCD::cnfCursorBlink  [private]
```

Cursor blinking configuration : 1 = cursor blink is on, 0 = cursor blink is off

Definition at line 313 of file LCD.h.

3.7.4.3 cnfCursorOnOff

```
bool LCD::cnfCursorOnOff  [private]
```

Cursor configuration : 1 = cursor on, 0 = cursor off

Definition at line 312 of file LCD.h.

3.7.4.4 cnfDisplayOnOff

```
bool LCD::cnfDisplayOnOff  [private]
```

Display configuration : 1 = display on, 0 = display off

Definition at line 311 of file LCD.h.

3.7.4.5 cnfEntryModeDir

```
bool LCD::cnfEntryModeDir  [private]
```

Entry mode direction configuration : 1 = cursor moves to right when DDRAM address is incremented, 0 = cursor moves to left when DDRAM address is incremented

Definition at line 314 of file LCD.h.

3.7.4.6 cnfEntryModeShift

```
bool LCD::cnfEntryModeShift  [private]
```

Entry mode configuration : 0 = no display shift is performed after a DDRAM read, 1 = a shift is performed

Definition at line 315 of file LCD.h.

3.7.4.7 cnfFontType

```
bool LCD::cnfFontType  [private]
```

Font type configuration, 0 = 5∗8 dots, 1 = 5∗11 dots

Definition at line 310 of file LCD.h.

3.7.4.8 cnfLineNumber

```
bool LCD::cnfLineNumber  [private]
```

Display line number configuration, 0 = 1-line mode, 1 = 2-line mode

Definition at line 309 of file LCD.h.

3.7.4.9 ddram_addr

```
uint8_t LCD::ddram_addr  [private]
```

Screen DDRAM address

Definition at line 317 of file LCD.h.

The documentation for this class was generated from the following files:

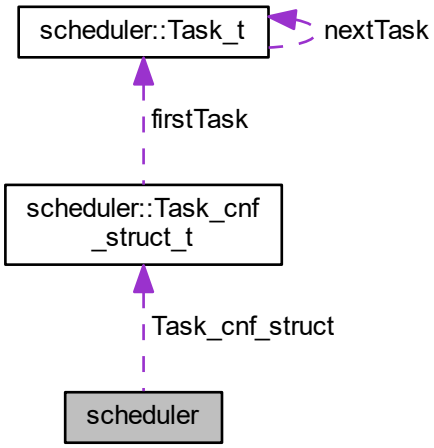
- LCD.h
- LCD.cpp

3.8 scheduler Class Reference

Scheduler class.

```
#include <scheduler.h>
```

Collaboration diagram for scheduler:



Classes

- struct Task_cnf_struct_t

  Task configuration structure.
- struct Task_t

  Type defining a task structure.

Public Member Functions

- scheduler ()

  scheduler class constructor
- void launchPeriodicTasks ()

  Main scheduler function.
- void startScheduling ()

  Starts the tasks scheduling.
- void addPeriodicTask (TaskPtr_t task_ptr, uint16_t a_period)

  Add a task into the scheduler.
- bool removePeriodicTask (TaskPtr_t task_ptr)

  Remove a task from the scheduler.
- uint32_t getPitNumber ()

  Get function for PIT number.

Private Types

- typedef struct scheduler::Task_t Task_t

  Type defining a task structure.

Private Attributes

- Task_cnf_struct_t Task_cnf_struct
- uint32_t pit_number

3.8.1 Detailed Description

Scheduler class.

This class defines the scheduler of the system. It is called by the main interrupt and calls successively all applicative functions according to their recurrence time.

Definition at line 28 of file scheduler.h.

3.8.2 Member Typedef Documentation

3.8.2.1 Task_t

```
typedef struct scheduler::Task_t scheduler::Task_t  [private]
```

Type defining a task structure.

This structure defines a task. A task is defined by a function to call (defined by its pointer), an ID and a period.

3.8.3 Constructor & Destructor Documentation

3.8.3.1 scheduler()

```
scheduler::scheduler ( )
```

scheduler class constructor

This function initializes the class scheduler

Returns

Nothing

Definition at line 19 of file scheduler.cpp.

Here is the call graph for this function:

3.8.4 Member Function Documentation

3.8.4.1 addPeriodicTask()

```
void scheduler::addPeriodicTask (
            TaskPtr_t task_ptr,
            uint16_t a_period )
```

Add a task into the scheduler.

This function create a new task in the scheduler linked to the function task_ptr with a period a_period and an ID a_task_id

Parameters

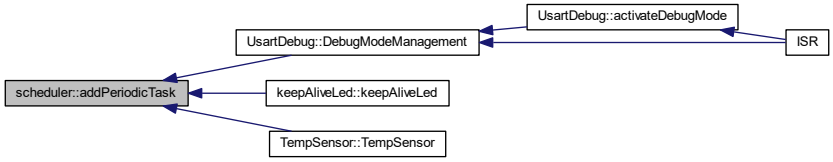
in	*task_ptr*	Pointer to the task which will be added
in	*a_period*	Period of the new task

Returns

Nothing

Definition at line 66 of file scheduler.cpp.

Here is the caller graph for this function:



3.8.4.2 getPitNumber()

```
uint32_t scheduler::getPitNumber ( )
```

Get function for PIT number.

This function returns the PIT number

Returns

PIT number

Definition at line 96 of file scheduler.cpp.

Here is the caller graph for this function:

3.8.4.3 launchPeriodicTasks()

```
void scheduler::launchPeriodicTasks ( )
```

Main scheduler function.

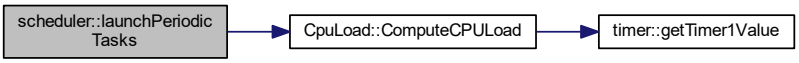
This function launches the scheduled tasks according to current software time and task configuration

Returns

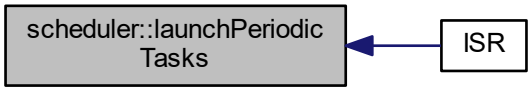
Nothing

Definition at line 32 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.8.4.4 removePeriodicTask()

```
bool scheduler::removePeriodicTask (
            TaskPtr_t task_ptr )
```

Remove a task from the scheduler.

This function finds the task defined by task_ptr in the scheduler and removes it.

Parameters

in	*task_ptr*	address of the task to remove from scheduler

Returns

TRUE if the task has been removed, FALSE if the task does not exist in the scheduler

Definition at line 102 of file scheduler.cpp.

Here is the caller graph for this function:

3.8.4.5 startScheduling()

```
void scheduler::startScheduling ( )
```

Starts the tasks scheduling.

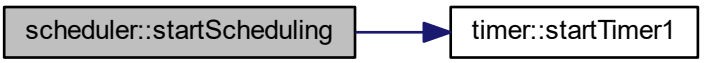
This function starts the timer which will trigger an interrupt every software period. When the interrupt is raised the scheduler will launch applications

Returns

Nothing

Definition at line 60 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.8.5 Member Data Documentation

3.8.5.1 pit_number

```
uint32_t scheduler::pit_number  [private]
```

Counter of periodic interrupts

Definition at line 115 of file scheduler.h.

3.8.5.2 Task_cnf_struct

```
Task_cnf_struct_t scheduler::Task_cnf_struct  [private]
```

Declaration of task configuration structure

Definition at line 113 of file scheduler.h.

The documentation for this class was generated from the following files:

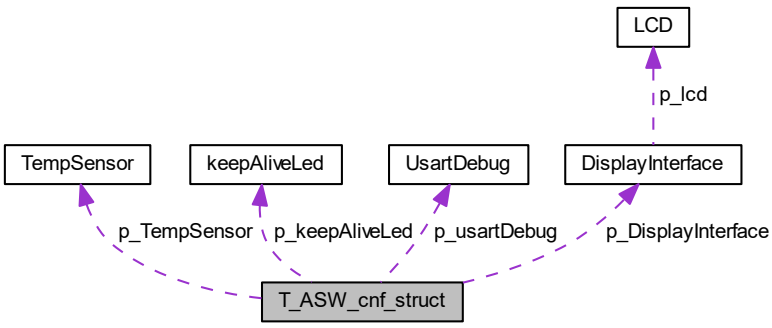
- scheduler.h
- scheduler.cpp

3.9 T_ASW_cnf_struct Struct Reference

ASW configuration structure.

```
#include <asw.h>
```

Collaboration diagram for T_ASW_cnf_struct:



Public Attributes

- UsartDebug * p_usartDebug
- keepAliveLed * p_keepAliveLed
- TempSensor * p_TempSensor
- DisplayInterface * p_DisplayInterface

3.9.1 Detailed Description

ASW configuration structure.

This structure contains all pointers to instanced applicative objects

Definition at line 24 of file asw.h.

3.9.2 Member Data Documentation

3.9.2.1 p_DisplayInterface

```
DisplayInterface* T_ASW_cnf_struct::p_DisplayInterface
```

Pointer to DisplayInterface object

Definition at line 29 of file asw.h.

3.9.2.2 p_keepAliveLed

```
keepAliveLed* T_ASW_cnf_struct::p_keepAliveLed
```

Pointer to keepAliveLed object

Definition at line 27 of file asw.h.

3.9.2.3 p_TempSensor

```
TempSensor* T_ASW_cnf_struct::p_TempSensor
```

Pointer to TempSensor object

Definition at line 28 of file asw.h.

3.9.2.4 p_usartDebug

```
UsartDebug* T_ASW_cnf_struct::p_usartDebug
```

Pointer to usart debug object

Definition at line 26 of file asw.h.

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

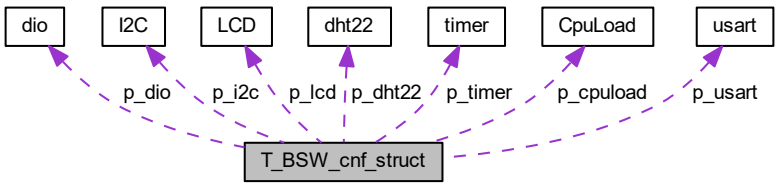
- asw.h

3.10 T_BSW_cnf_struct Struct Reference

BSW configuration structure.

```
#include <bsw.h>
```

Collaboration diagram for T_BSW_cnf_struct:



Public Attributes

- usart * p_usart
- dio * p_dio
- timer * p_timer
- dht22 * p_dht22
- CpuLoad * p_cpuload
- I2C * p_i2c
- LCD * p_lcd

3.10.1 Detailed Description

BSW configuration structure.

This structure contains all pointers to instanced drivers objects

Definition at line 33 of file bsw.h.

3.10.2 Member Data Documentation

3.10.2.1 p_cpuload

```
CpuLoad* T_BSW_cnf_struct::p_cpuload
```

Pointer to cpu load library object

Definition at line 39 of file bsw.h.

3.10.2.2 p_dht22

```
dht22* T_BSW_cnf_struct::p_dht22
```

Pointer to dht22 driver object

Definition at line 38 of file bsw.h.

3.10.2.3 p_dio

```
dio* T_BSW_cnf_struct::p_dio
```

Pointer to dio driver object

Definition at line 36 of file bsw.h.

3.10.2.4 p_i2c

```
I2C* T_BSW_cnf_struct::p_i2c
```

Pointer to I2C driver object

Definition at line 40 of file bsw.h.

3.10.2.5 p_lcd

```
LCD* T_BSW_cnf_struct::p_lcd
```

Pointer to LCD driver object

Definition at line 41 of file bsw.h.

3.10.2.6 p_timer

```
timer* T_BSW_cnf_struct::p_timer
```

Pointer to timer driver object

Definition at line 37 of file bsw.h.

3.10.2.7 p_usart

```
usart* T_BSW_cnf_struct::p_usart
```

Pointer to usart driver object

Definition at line 35 of file bsw.h.

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

- bsw.h

3.11 T_LCD_conf_struct Struct Reference

```
#include <LCD.h>
```

Public Attributes

- bool backlight_en
- bool lineNumber_cnf
- bool fontType_cnf
- bool display_en
- bool cursor_en
- bool cursorBlink_en
- bool entryModeDir
- bool entryModeShift

3.11.1 Detailed Description

Definition at line 127 of file LCD.h.

3.11.2 Member Data Documentation

3.11.2.1 backlight_en

```
bool T_LCD_conf_struct::backlight_en
```

Definition at line 129 of file LCD.h.

3.11.2.2 cursor_en

```
bool T_LCD_conf_struct::cursor_en
```

Definition at line 133 of file LCD.h.

3.11.2.3 cursorBlink_en

```
bool T_LCD_conf_struct::cursorBlink_en
```

Definition at line 134 of file LCD.h.

3.11.2.4 display_en

```
bool T_LCD_conf_struct::display_en
```

Definition at line 132 of file LCD.h.

3.11.2.5 entryModeDir

```
bool T_LCD_conf_struct::entryModeDir
```

Definition at line 135 of file LCD.h.

3.11.2.6 entryModeShift

```
bool T_LCD_conf_struct::entryModeShift
```

Definition at line 136 of file LCD.h.

3.11.2.7 fontType_cnf

```
bool T_LCD_conf_struct::fontType_cnf
```

Definition at line 131 of file LCD.h.

3.11.2.8 lineNumber_cnf

```
bool T_LCD_conf_struct::lineNumber_cnf
```

Definition at line 130 of file LCD.h.

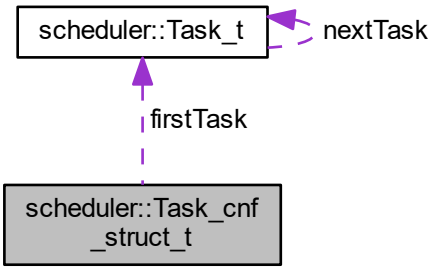
The documentation for this struct was generated from the following file:

- LCD.h

3.12 scheduler::Task_cnf_struct_t Struct Reference

Task configuration structure.

Collaboration diagram for scheduler::Task_cnf_struct_t:



Public Attributes

- Task_t * firstTask
- uint8_t task_nb

3.12.1 Detailed Description

Task configuration structure.

This structure contains task list and memorizes the number of current tasks launched

Definition at line 106 of file scheduler.h.

3.12.2 Member Data Documentation

3.12.2.1 firstTask

```
Task_t* scheduler::Task_cnf_struct_t::firstTask
```

Pointer to the first task to launch

Definition at line 108 of file scheduler.h.

3.12.2.2 task_nb

```
uint8_t scheduler::Task_cnf_struct_t::task_nb
```

Number of task

Definition at line 109 of file scheduler.h.

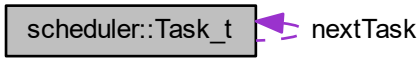
The documentation for this struct was generated from the following file:

- scheduler.h

3.13 scheduler::Task_t Struct Reference

Type defining a task structure.

Collaboration diagram for scheduler::Task_t:



Public Attributes

- TaskPtr_t TaskPtr
- uint16_t period
- Task_t * nextTask

3.13.1 Detailed Description

Type defining a task structure.

This structure defines a task. A task is defined by a function to call (defined by its pointer), an ID and a period.

Definition at line 94 of file scheduler.h.

3.13.2 Member Data Documentation

3.13.2.1 nextTask

```
Task_t* scheduler::Task_t::nextTask
```

Pointer to the next task to launch

Definition at line 98 of file scheduler.h.

3.13.2.2 period

```
uint16_t scheduler::Task_t::period
```

Period of the task

Definition at line 97 of file scheduler.h.

3.13.2.3 TaskPtr

```
TaskPtr_t scheduler::Task_t::TaskPtr
```

Pointer to the task

Definition at line 96 of file scheduler.h.

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

- scheduler.h

3.14 TempSensor Class Reference

Class for temperature sensor.

```
#include <TempSensor.h>
```

Public Member Functions

- TempSensor ()

  Class constructor.
- uint16_t * getTempPtr ()

  Get pointer to data raw_temperature.
- uint16_t * getHumPtr ()

  Get pointer to data raw_humidity.
- bool getTemp (uint16_t *temp)

  Get temperature data.
- bool getHumidity (uint16_t *hum)

  Get humidity data.
- void setValidity (bool validity)

  Set data val_validity.
- void updateLastValidValues ()

Static Public Member Functions

- static void readTempSensor_task ()

  Task for reading temperature and humidity values.

Private Attributes

- uint16_t read_temperature
- uint16_t read_humidity
- bool validity_last_read
- bool validity
- uint32_t valid_pit
- uint16_t valid_temp
- uint16_t valid_hum

3.14.1 Detailed Description

Class for temperature sensor.

This class defines all functions used to read data from temperature sensor and monotor it

Definition at line 19 of file TempSensor.h.

3.14.2 Constructor & Destructor Documentation

3.14.2.1 TempSensor()

```
TempSensor::TempSensor ( )
```

Class constructor.

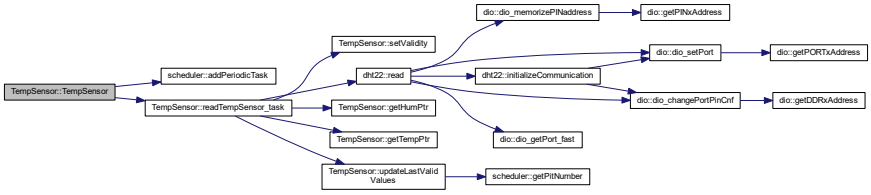
This function initializes all data of the class TempSensor

Returns

Nothing

Definition at line 16 of file TempSensor.cpp.

Here is the call graph for this function:



3.14.3 Member Function Documentation

3.14.3.1 getHumidity()

```
bool TempSensor::getHumidity (
            uint16_t * hum )
```

Get humidity data.

This function returns the value of the humidity. If the official value is not valid, the function return false.

Parameters

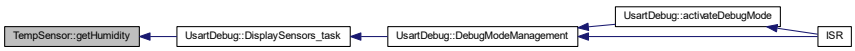
out	*hum*	Humidity value

Returns

Validity of humidity

Definition at line 66 of file TempSensor.cpp.

Here is the caller graph for this function:



3.14.3.2 getHumPtr()

```
uint16_t * TempSensor::getHumPtr ( )
```

Get pointer to data raw_humidity.

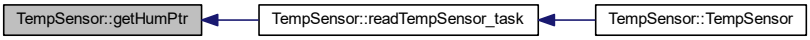
This function returns a pointer to the class member raw_humidity

Returns

Pointer to raw_humidity

Definition at line 41 of file TempSensor.cpp.

Here is the caller graph for this function:



3.14.3.3 getTemp()

```
bool TempSensor::getTemp (
            uint16_t * temp )
```

Get temperature data.

This function returns the value of the temperature. If the official value is not valid, the function return false.

Parameters

out	*temp*	Temperature value

Returns

Validity of temperature

Definition at line 72 of file TempSensor.cpp.

Here is the caller graph for this function:

3.14.3.4 getTempPtr()

```
uint16_t * TempSensor::getTempPtr ( )
```

Get pointer to data raw_temperature.

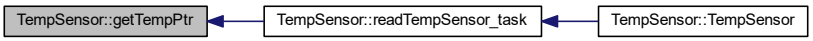
This function returns a pointer to the class member raw_temperature

Returns

Pointer to raw_temperature

Definition at line 46 of file TempSensor.cpp.

Here is the caller graph for this function:



3.14.3.5 readTempSensor_task()

```
void TempSensor::readTempSensor_task ( )  [static]
```

Task for reading temperature and humidity values.

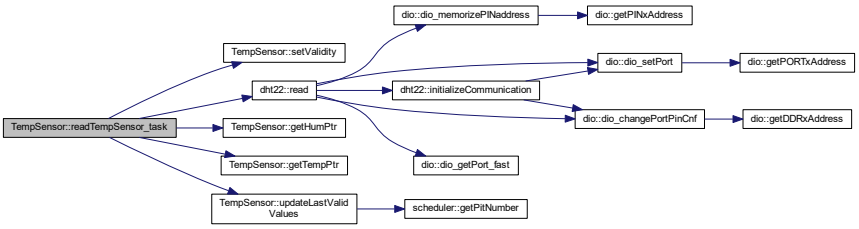
This task reads temperature and humidity data using DHT22 driver. It is called every 5 seconds.

Returns

Nothing

Definition at line 30 of file TempSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.14.3.6 setValidity()

```
void TempSensor::setValidity (
            bool validity )
```

Set data val_validity.

This function sets the class member val_validity

Parameters

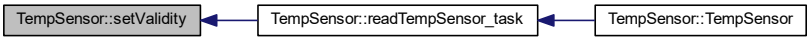
in	*validity*	Value of validity

Returns

Nothing

Definition at line 36 of file TempSensor.cpp.

Here is the caller graph for this function:

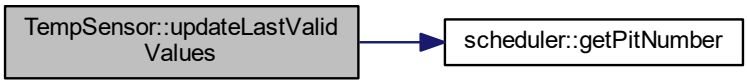


3.14.3.7 updateLastValidValues()

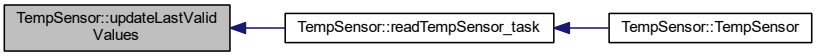
```
void TempSensor::updateLastValidValues ( )
```

Definition at line 51 of file TempSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.14.4 Member Data Documentation

3.14.4.1 read_humidity

```
uint16_t TempSensor::read_humidity  [private]
```

Raw value of humidity read from DHT22 (= real humidity ∗10)

Definition at line 86 of file TempSensor.h.

3.14.4.2 read_temperature

```
uint16_t TempSensor::read_temperature  [private]
```

Raw value of temperature read from DHT22 (= real temperature ∗10)

Definition at line 85 of file TempSensor.h.

3.14.4.3 valid_hum

```
uint16_t TempSensor::valid_hum  [private]
```

Valid value of humidity

Definition at line 93 of file TempSensor.h.

3.14.4.4 valid_pit

```
uint32_t TempSensor::valid_pit  [private]
```

pit number of the last time when data were valid

Definition at line 90 of file TempSensor.h.

3.14.4.5 valid_temp

```
uint16_t TempSensor::valid_temp  [private]
```

Valid value of temperature

Definition at line 92 of file TempSensor.h.

3.14.4.6 validity

```
bool TempSensor::validity  [private]
```

validity of official temperature and humidity data

Definition at line 89 of file TempSensor.h.

3.14.4.7 validity_last_read

```
bool TempSensor::validity_last_read  [private]
```

Validity of last read temperature and humidity data

Definition at line 87 of file TempSensor.h.

The documentation for this class was generated from the following files:

- TempSensor.h
- TempSensor.cpp

3.15 timer Class Reference

Class defining a timer.

```
#include <timer.h>
```

Public Member Functions

- timer ()
  Class constructor.
- void configureTimer1 (uint16_t a_prescaler, uint16_t a_ctcValue)
  Configures Timer #1.
- void startTimer1 ()
  Start Timer #1.
- void stopTimer1 ()
  Stops Timer #1.
- uint16_t getTimer1Value ()
  Reads current value of timer #1.

Private Attributes

- uint8_t prescaler

3.15.1 Detailed Description

Class defining a timer.

This class defines a timer/counter. The selected timer is configured in CTC mode and interrupts are enabled. The prescaler value and CTC value can both be configured by user.

Definition at line 22 of file timer.h.

3.15.2 Constructor & Destructor Documentation

3.15.2.1 timer()

```
timer::timer ( )
```

Class constructor.

This function initializes class attributes

Returns

Nothing

Definition at line 13 of file timer.cpp.

3.15.3 Member Function Documentation

3.15.3.1 configureTimer1()

```
void timer::configureTimer1 (
            uint16_t a_prescaler,
            uint16_t a_ctcValue )
```

Configures Timer #1.

This function configures hardware timer #1 in CTC mode, enables its interrupts, sets prescaler to a_prescaler and CTC value to a_ctcValue

Parameters

in	*a_prescaler*	prescaler value
in	*a_ctcValue*	Value to which the counter will compare before raising an interrupt

Returns

Nothing

Definition at line 18 of file timer.cpp.

Here is the caller graph for this function:

3.15.3.2 getTimer1Value()

```
uint16_t timer::getTimer1Value ( )  [inline]
```

Reads current value of timer #1.

This function reads the value of of timer #1 using register TCNT1. The function is inlined to speed up SW execution.

Returns

Current timer value

Definition at line 61 of file timer.h.

Here is the caller graph for this function:

3.15.3.3 startTimer1()

```
void timer::startTimer1 ( )
```

Start Timer #1.

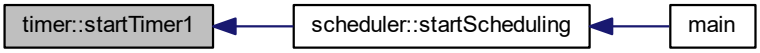
This functions starts Timer #1. Timer shall be initialized before this function is called.

Returns

Nothing

Definition at line 56 of file timer.cpp.

Here is the caller graph for this function:



3.15.3.4 stopTimer1()

```
void timer::stopTimer1 ( )
```

Stops Timer #1.

This functions stops timer #1 by resetting bits 0-2 of TCCR1B

Returns

Nothing

Definition at line 67 of file timer.cpp.

3.15.4 Member Data Documentation

3.15.4.1 prescaler

```
uint8_t timer::prescaler  [private]
```

Definition at line 67 of file timer.h.

The documentation for this class was generated from the following files:

- timer.h
- timer.cpp

3.16 usart Class Reference

USART serial bus class.

```
#include <usart.h>
```

Public Member Functions

- usart (uint16_t a_BaudRate)
  Class usart constructor.
- void usart_sendString (uint8_t *str)
  Sending a string on USART link.
- void setBaudRate (uint16_t a_BaudRate)
  Setting baud rate.
- void usart_init ()
  USART hardware initialization.
- uint8_t usart_read ()
  USART read function.

Static Private Member Functions

- static void usart_transmit (uint8_t Data)
  USART Transmit data.

Private Attributes

- uint16_t BaudRate

3.16.1 Detailed Description

USART serial bus class.

This class defines all useful functions for USART serial bus

Definition at line 16 of file usart.h.

3.16.2 Constructor & Destructor Documentation

3.16.2.1 usart()

```
usart::usart (
            uint16_t a_BaudRate )
```

Class usart constructor.

Initializes the class and call hardware initialization function

Parameters

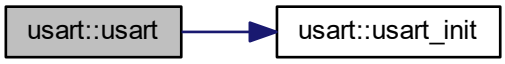
in	*a_BaudRate*	Desired Baud Rate (16 bit) - up to 57600

Returns

Nothing.

Definition at line 14 of file usart.cpp.

Here is the call graph for this function:



3.16.3 Member Function Documentation

3.16.3.1 setBaudRate()

```
void usart::setBaudRate (
            uint16_t a_BaudRate )  [inline]
```

Setting baud rate.

This function sets the attribute BaudRate of the class usart

Parameters

in	*a_BaudRate*	Desired Baud Rate (16 bit) - up to 57600

Returns

Nothing

Definition at line 63 of file usart.cpp.

3.16.3.2 usart_init()

```
void usart::usart_init ( )
```

USART hardware initialization.

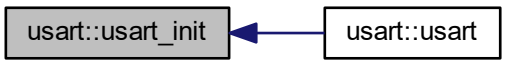
This function will initialize the USART using selected baudrate. User must pay attention to select one of the usually used Baud Rate (9600, 19200, 38400, 57600). Note that since an uint16 is used as argument, Baud rate cannot be more than 57600.

Returns

Nothing.

Definition at line 21 of file usart.cpp.

Here is the caller graph for this function:



3.16.3.3 usart_read()

```
uint8_t usart::usart_read ( )
```

USART read function.

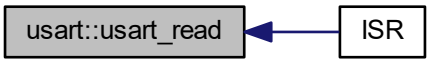
This function will read reception register of USART

Returns

The function returns the 8 bits read from reception buffer

Definition at line 79 of file usart.cpp.

Here is the caller graph for this function:



3.16.3.4 usart_sendString()

```
void usart::usart_sendString (
            uint8_t * str )
```

Sending a string on USART link.

Just write data to the Serial link using usart_trabsmit function

Parameters

in	*str*	Pointer to the string being sent

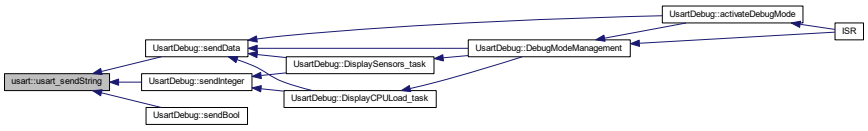
Returns

Nothing.

Definition at line 44 of file usart.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



3.16.3.5 usart_transmit()

```
void usart::usart_transmit (
            uint8_t Data )  [static], [private]
```

USART Transmit data.

Nothing Special. It just wait for the transmit buffer is empty before writing it again.

Parameters

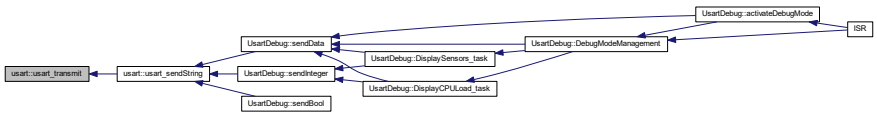
in	*Data*	Desired data char to transmit

Returns

Nothing.

Definition at line 70 of file usart.cpp.

Here is the caller graph for this function:



3.16.4 Member Data Documentation

3.16.4.1 BaudRate

```
uint16_t usart::BaudRate  [private]
```

Defines the baud rate used by driver

Definition at line 70 of file usart.h.

The documentation for this class was generated from the following files:

- usart.h
- usart.cpp

3.17 UsartDebug Class Reference

Class used for debugging on usart link.

```
#include <debug.h>
```

Public Member Functions

- UsartDebug ()
  Class UsartDebug constructor.
- void sendInteger (uint16_t data, uint8_t base)
  Send a integer data on USART link.
- void sendBool (bool data)
  Send a boolean data on USART link.
- bool isDebugModeActive ()
  Check is debug mode is active or not.
- void activateDebugMode ()
  Activates debug mode.
- void DebugModeManagement (uint8_t rcv_char)
  Management of debug mode.

Static Public Member Functions

- static void DisplaySensors_task ()
  Displays sensors data on usart link.
- static void DisplayCPULoad_task ()
  Displays CPU load data on usart link.

Private Member Functions

- void sendData (char *str)
  Send a string on USART link.

Private Attributes

- debug_state_t debug_state
- bool debugModeActive_F

3.17.1 Detailed Description

Class used for debugging on usart link.

This class defines functions used for sending debug data on USART link.

Definition at line 33 of file debug.h.

3.17.2 Constructor & Destructor Documentation

3.17.2.1 UsartDebug()

```
UsartDebug::UsartDebug ( )
```

Class UsartDebug constructor.

Initializes the class UsartDebug

Returns

Nothing

Definition at line 31 of file debug.cpp.

3.17.3 Member Function Documentation

3.17.3.1 activateDebugMode()

```
void UsartDebug::activateDebugMode ( )
```

Activates debug mode.

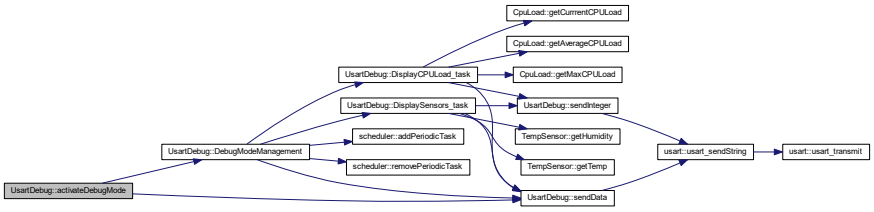
This function activates USART debug mode.

Returns

Nothing

Definition at line 126 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.17.3.2 DebugModeManagement()

```
void UsartDebug::DebugModeManagement (
            uint8_t rcv_char )
```

Management of debug mode.

This function manages the debug mode according to the following state machine :

- init state : display main menu
- WAIT_INIT state : handles user choice in main menu and selects next state
- DISPLAY_DATA state : display sensor data periodically
- DISPLAY CPU LOAD : display CPU load periodically

It is called each time a data is received on USART and debug mode is active

Parameters

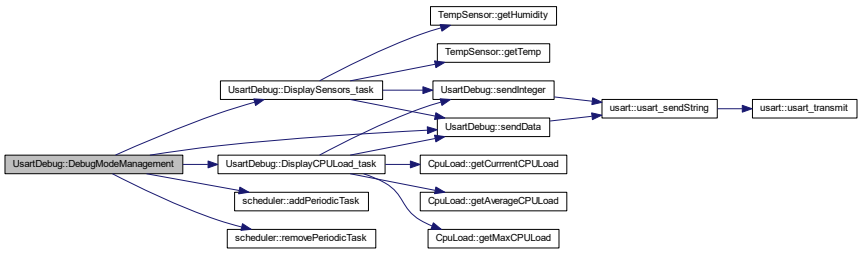
in	*rcv_char*	8 bits character received on USART

Returns

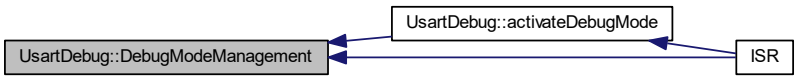
Nothing

Definition at line 134 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.17.3.3 DisplayCPULoad_task()

```
void UsartDebug::DisplayCPULoad_task ( )  [static]
```

Displays CPU load data on usart link.

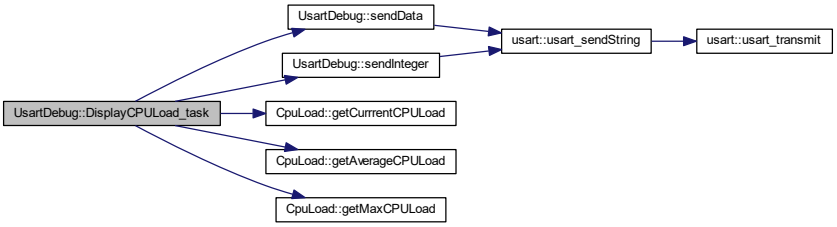
This task sends CPU load data (current and average load) on usart link every 5 seconds

Returns

Nothing

Definition at line 110 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.17.3.4 DisplaySensors_task()

```
void UsartDebug::DisplaySensors_task ( )  [static]
```

Displays sensors data on usart link.

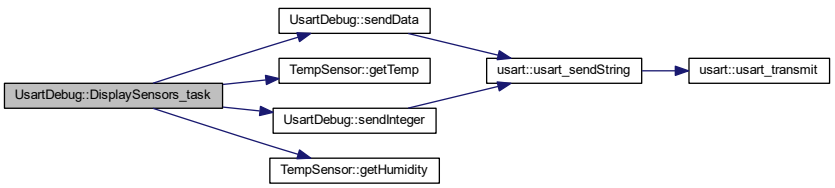
This task sends sensors data (temperature and humidity) on usart link every 5 seconds

Returns

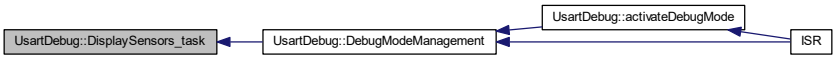
Nothing

Definition at line 69 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.17.3.5 isDebugModeActive()

```
bool UsartDebug::isDebugModeActive ( )
```

Check is debug mode is active or not.

This function checks if debug mode is active or not.

Returns

TRUE is debug mode is active, FALSE otherwise

Definition at line 121 of file debug.cpp.

Here is the caller graph for this function:

3.17.3.6 sendBool()

```
void UsartDebug::sendBool (
            bool data )
```

Send a boolean data on USART link.

This functions sends the requested boolean on USART link by calling driver's transmission function. The boolean data is first converted into a string and then sent

Parameters

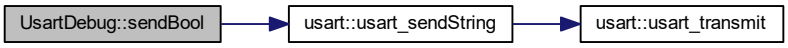
in	*data*	boolean data to be sent

Returns

Nothing

Definition at line 57 of file debug.cpp.

Here is the call graph for this function:



3.17.3.7 sendData()

```
void UsartDebug::sendData (
            char * str )  [private]
```

Send a string on USART link.

This functions sends the requested string on USART link by calling driver's transmission function

Parameters

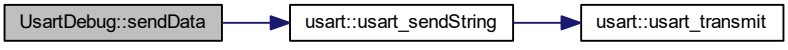
in	*str*	Pointer to the string being sent

Returns

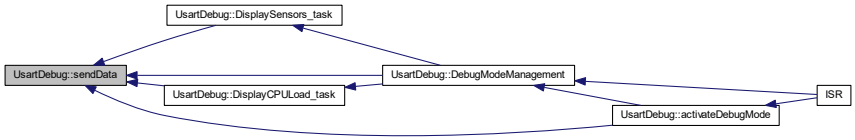
Nothing

Definition at line 37 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.17.3.8 sendInteger()

```
void UsartDebug::sendInteger (
            uint16_t data,
            uint8_t base )
```

Send a integer data on USART link.

This functions sends the requested integer on USART link by calling driver's transmission function. The integer is first converted into a string and then sent

Parameters

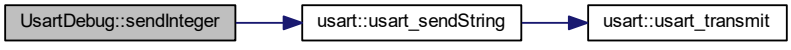
in	*data*	integer data to be sent
in	*base*	numerical base used to convert integer into string (between 2 and 36)

Returns

Nothing

Definition at line 43 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.17.4 Member Data Documentation

3.17.4.1 debug_state

```
debug_state_t UsartDebug::debug_state  [private]
```

Definition at line 116 of file debug.h.

3.17.4.2 debugModeActive_F

```
bool UsartDebug::debugModeActive_F  [private]
```

Debug mode activation flag

Definition at line 117 of file debug.h.

The documentation for this class was generated from the following files:

- debug.h
- debug.cpp

Chapter 4

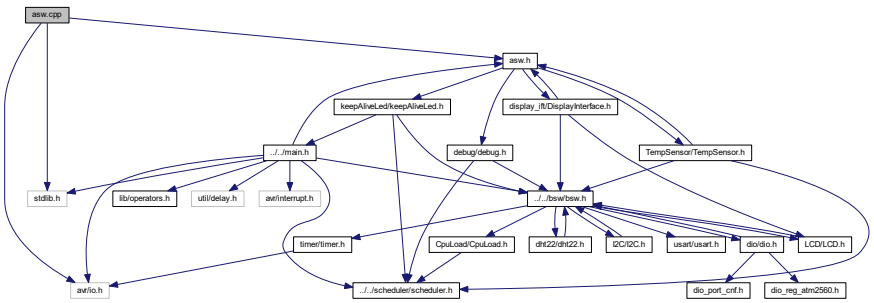
File Documentation

4.1 asw.cpp File Reference

ASW main file.

```
#include <stdlib.h>
#include <avr/io.h>
#include "asw.h"
```

Include dependency graph for asw.cpp:



Functions

- void asw_init ()

  Initialization of ASW.

Variables

- T_ASW_cnf_struct ASW_cnf_struct

4.1.1 Detailed Description

ASW main file.

Date

15 mars 2018

Author

nicls67

4.1.2 Function Documentation

4.1.2.1 asw_init()

```
void asw_init ( )
```

Initialization of ASW.

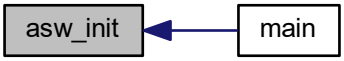
This function instantiates all applicative objects. The addresses of objects are then stored in ASW_cnf_struct structure. This function shall be called after BSW initialization function.

Returns

Nothing

Definition at line 19 of file asw.cpp.

Here is the caller graph for this function:



4.1.3 Variable Documentation

4.1.3.1 ASW_cnf_struct

```
T_ASW_cnf_struct ASW_cnf_struct
```

ASW configuration structure

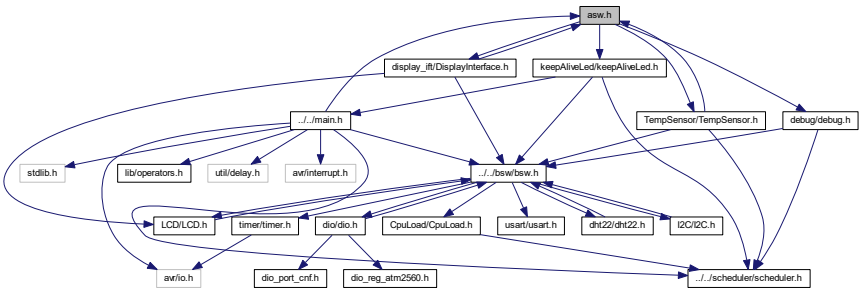
Definition at line 16 of file asw.cpp.

4.2 asw.h File Reference

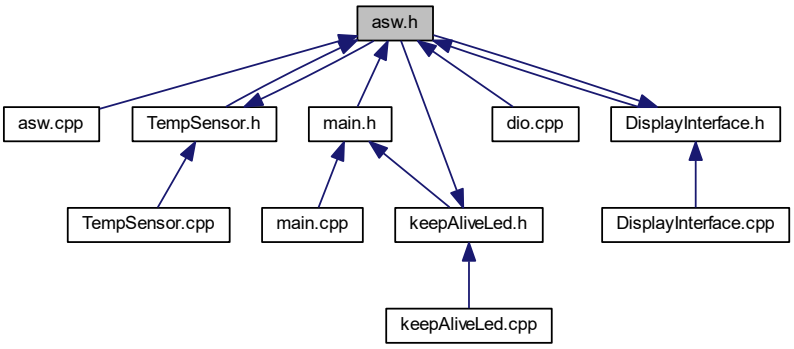
ASW main header file.

```
#include "debug/debug.h"
#include "keepAliveLed/keepAliveLed.h"
#include "TempSensor/TempSensor.h"
#include "display_ift/DisplayInterface.h"
```

Include dependency graph for asw.h:



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



Classes

- struct T_ASW_cnf_struct

  ASW configuration structure.

Functions

- void asw_init ()

  Initialization of ASW.

Variables

- T_ASW_cnf_struct ASW_cnf_struct

4.2.1 Detailed Description

ASW main header file.

Date

15 mars 2018

Author

nicls67

4.2.2 Function Documentation

4.2.2.1 asw_init()

```
void asw_init ( )
```

Initialization of ASW.

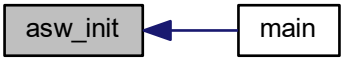
This function instantiates all applicative objects. The addresses of objects are then stored in ASW_cnf_struct structure. This function shall be called after BSW initialization function.

Returns

Nothing

Definition at line 19 of file asw.cpp.

Here is the caller graph for this function:



4.2.3 Variable Documentation

4.2.3.1 ASW_cnf_struct

```
T_ASW_cnf_struct ASW_cnf_struct
```

ASW configuration structure

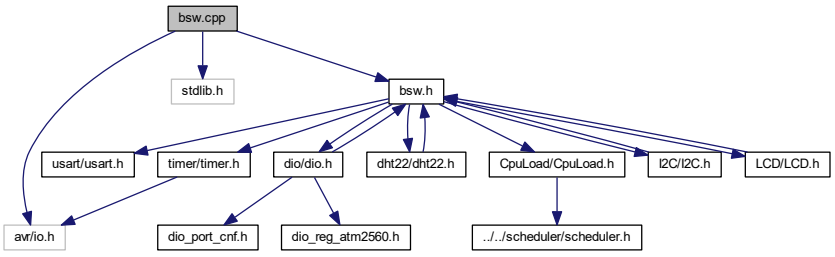
Definition at line 16 of file asw.cpp.

4.3 bsw.cpp File Reference

BSW main file.

```
#include <avr/io.h>
#include <stdlib.h>
#include "bsw.h"
```

Include dependency graph for bsw.cpp:



Functions

- void bsw_init ()
  Initialization of BSW.

Variables

- T_BSW_cnf_struct BSW_cnf_struct

4.3.1 Detailed Description

BSW main file.

Date

13 mars 2018

Author

nicls67

4.3.2 Function Documentation

4.3.2.1 bsw_init()

```
void bsw_init ( )
```

Initialization of BSW.

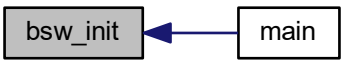
This function instantiates all driver objects, leading hardware initialization. The addresses of driver objects are then stored in BSW_cnf_struct structure.

Returns

Nothing

Definition at line 18 of file bsw.cpp.

Here is the caller graph for this function:



4.3.3 Variable Documentation

4.3.3.1 BSW_cnf_struct

```
T_BSW_cnf_struct BSW_cnf_struct
```

BSW configuration structure

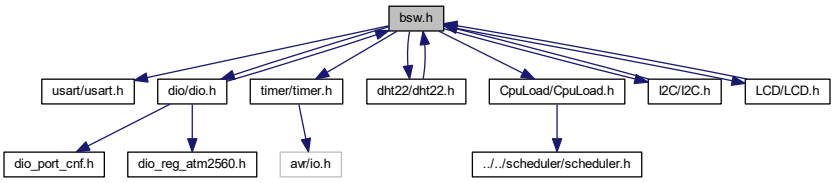
Definition at line 16 of file bsw.cpp.

4.4 bsw.h File Reference

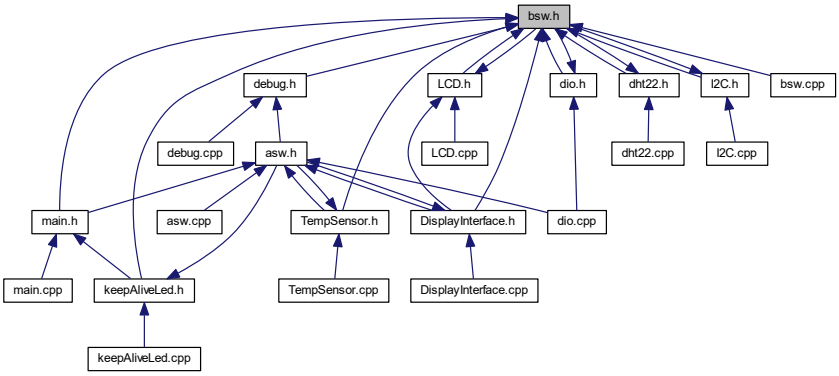
BSW main header file.

```
#include "usart/usart.h"
#include "dio/dio.h"
#include "timer/timer.h"
#include "dht22/dht22.h"
#include "CpuLoad/CpuLoad.h"
#include "I2C/I2C.h"
#include "LCD/LCD.h"
```

Include dependency graph for bsw.h:



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



Classes

- struct T_BSW_cnf_struct

  BSW configuration structure.

Macros

- #define USART_BAUDRATE (uint16_t)9600
- #define I2C_BITRATE (uint32_t)100000

Functions

- void bsw_init ()
  Initialization of BSW.

Variables

- T_BSW_cnf_struct BSW_cnf_struct

4.4.1 Detailed Description

BSW main header file.

Date

13 mars 2018

Author

nicls67

4.4.2 Macro Definition Documentation

4.4.2.1 I2C_BITRATE

```
#define I2C_BITRATE (uint32_t)100000
```

I2C bus bitrate is 100 kHz

Definition at line 27 of file bsw.h.

4.4.2.2 USART_BAUDRATE

```
#define USART_BAUDRATE (uint16_t)9600
```

usart connection to PC uses a baud rate of 9600

Definition at line 26 of file bsw.h.

4.4.3 Function Documentation

4.4.3.1 bsw_init()

```
void bsw_init ( )
```

Initialization of BSW.

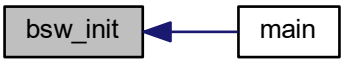
This function instantiates all driver objects, leading hardware initialization. The addresses of driver objects are then stored in BSW_cnf_struct structure.

Returns

Nothing

Definition at line 18 of file bsw.cpp.

Here is the caller graph for this function:



4.4.4 Variable Documentation

4.4.4.1 BSW_cnf_struct

```
T_BSW_cnf_struct BSW_cnf_struct
```

BSW configuration structure

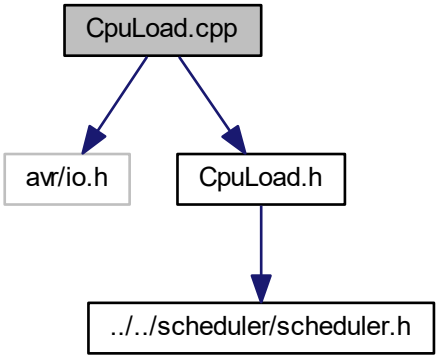
Definition at line 16 of file bsw.cpp.

4.5 CpuLoad.cpp File Reference

Defines functions of class CpuLoad.

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

Include dependency graph for CpuLoad.cpp:



4.5.1 Detailed Description

Defines functions of class CpuLoad.

Date

21 mars 2019

Author

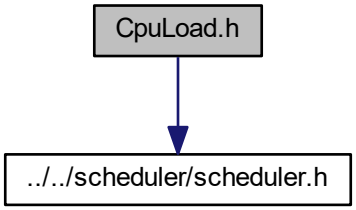
nicls67

4.6 CpuLoad.h File Reference

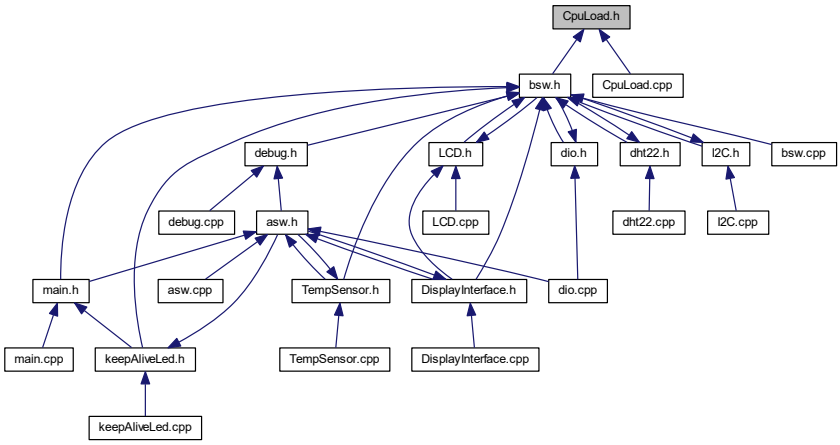
CpuLoad class header file.

```
#include "../../scheduler/scheduler.h"
```

Include dependency graph for CpuLoad.h:



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



Classes

- class CpuLoad

  Class defining CPU load libraries.

Macros

- #define NB_OF_SAMPLES 50

4.6.1 Detailed Description

CpuLoad class header file.

Date

21 mars 2019

Author

nicls67

4.6.2 Macro Definition Documentation

4.6.2.1 NB_OF_SAMPLES

```
#define NB_OF_SAMPLES 50
```

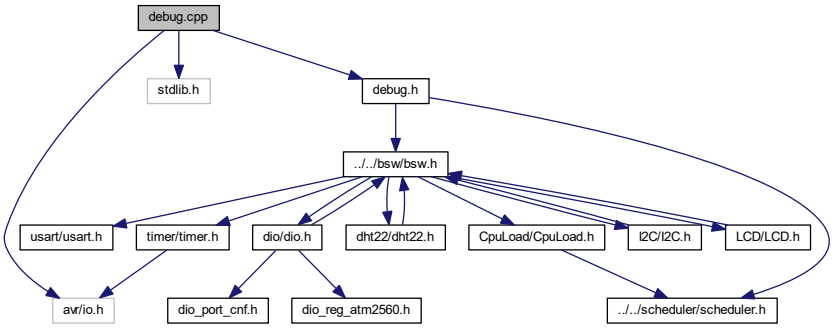
Definition at line 13 of file CpuLoad.h.

4.7 debug.cpp File Reference

This file defines classes for log and debug data transmission on USART link.

```
#include <avr/io.h>
#include <stdlib.h>
#include "debug.h"
```

Include dependency graph for debug.cpp:



Variables

- const char str_debug_main_menu [ ]

  Main menu of debug mode.

4.7.1 Detailed Description

This file defines classes for log and debug data transmission on USART link.

Date

15 mars 2018

Author

nicls67

4.7.2 Variable Documentation

4.7.2.1 str_debug_main_menu

```
const char str_debug_main_menu[]
```

Initial value:

```
=
        "\n\n"
        "Menu principal :  \n"
        "1 : Afficher donnees capteurs\n"
        "2 : Afficher charge CPU\n"
        "\n"
        "s : Quitter debug\n"
```

Main menu of debug mode.

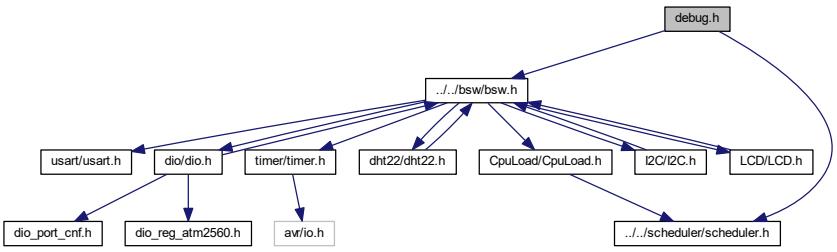
Definition at line 20 of file debug.cpp.

4.8 debug.h File Reference

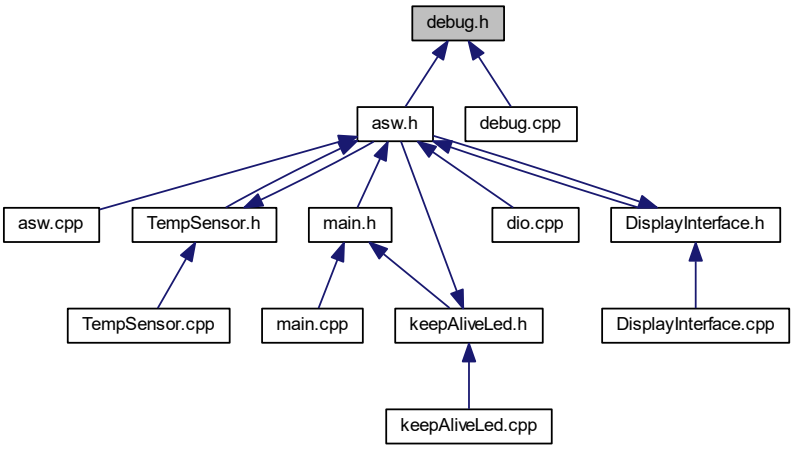
Header file for debug and logging functions.

```
#include "../../bsw/bsw.h"
#include "../../scheduler/scheduler.h"
```

Include dependency graph for debug.h:



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



Classes

- class UsartDebug

  Class used for debugging on usart link.

Macros

- #define PERIOD_MS_TASK_DISPLAY_SENSORS 5000
- #define PERIOD_MS_TASK_DISPLAY_CPU_LOAD 5000

Enumerations

- enum debug_state_t { INIT, WAIT_INIT, DISPLAY_DATA, DISPLAY_CPU_LOAD }

  Defines the debug states.

4.8.1 Detailed Description

Header file for debug and logging functions.

Date

15 mars 2018

Author

nicls67

4.8.2 Macro Definition Documentation

4.8.2.1 PERIOD_MS_TASK_DISPLAY_CPU_LOAD

```
#define PERIOD_MS_TASK_DISPLAY_CPU_LOAD 5000
```

Period for displaying CPU load data

Definition at line 14 of file debug.h.

4.8.2.2 PERIOD_MS_TASK_DISPLAY_SENSORS

```
#define PERIOD_MS_TASK_DISPLAY_SENSORS 5000
```

Period for displaying temperature and humidity data

Definition at line 13 of file debug.h.

4.8.3 Enumeration Type Documentation

4.8.3.1 debug_state_t

```
enum debug_state_t
```

Defines the debug states.

Enumerator

INIT	Init state : display the main menu
WAIT_INIT	Wait for a received character in init state
DISPLAY_DATA	Display sensor data in continuous
DISPLAY_CPU_LOAD	Display CPU load in continuous

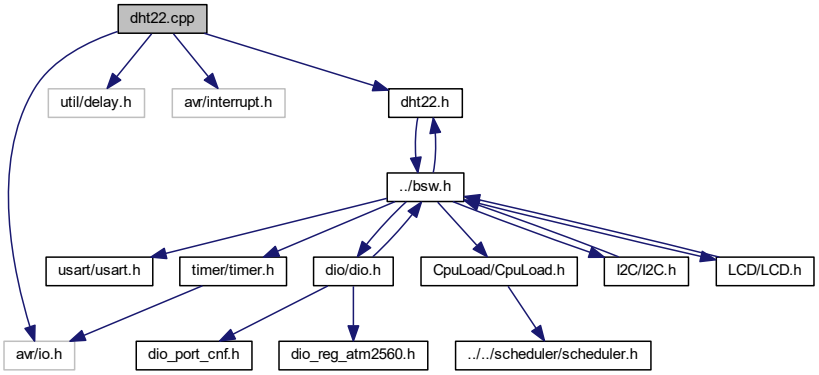
Definition at line 20 of file debug.h.

4.9 dht22.cpp File Reference

This file defines classes for DHT22 driver.

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "dht22.h"
```

Include dependency graph for dht22.cpp:



Macros

- #define MAX_WAIT_TIME_US 100

4.9.1 Detailed Description

This file defines classes for DHT22 driver.

Date

23 mars 2018

Author

nicls67

4.9.2 Macro Definition Documentation

4.9.2.1 MAX_WAIT_TIME_US

```
#define MAX_WAIT_TIME_US 100
```

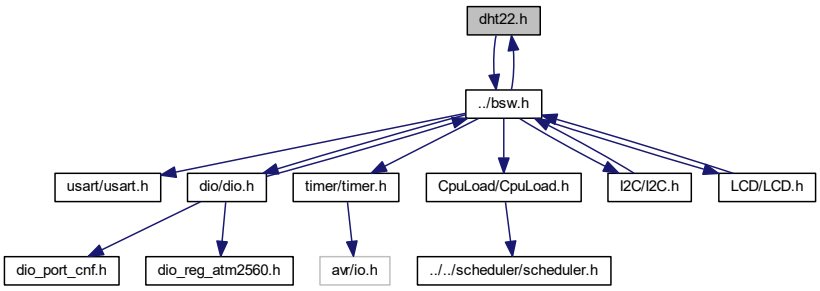
Definition at line 20 of file dht22.cpp.

4.10 dht22.h File Reference

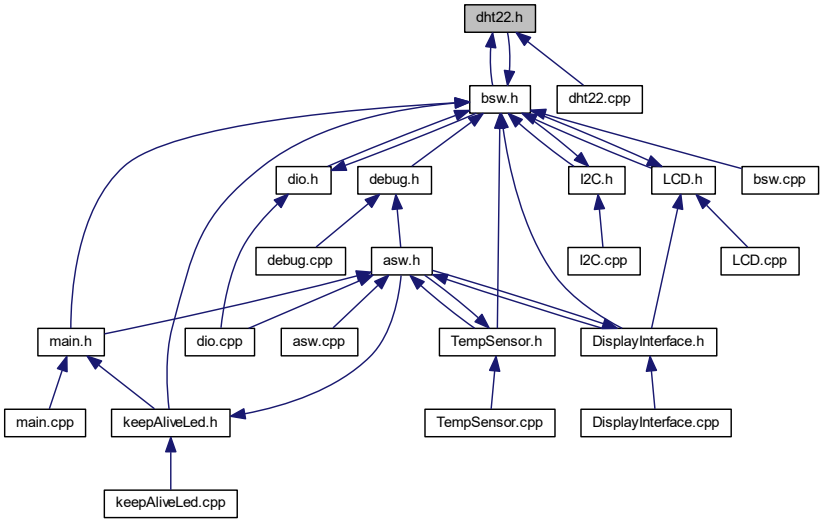
DHT22 driver header file.

```
#include "../bsw.h"
```

Include dependency graph for dht22.h:



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



Classes

- class dht22

  DHT 22 driver class.

Macros

- #define DHT22_PORT ENCODE_PORT(PORT_B, 6)

4.10.1 Detailed Description

DHT22 driver header file.

Date

23 mars 2018

Author

nicls67

4.10.2 Macro Definition Documentation

4.10.2.1 DHT22_PORT

```
#define DHT22_PORT ENCODE_PORT(PORT_B, 6)
```

DHT22 is connected to port PB6

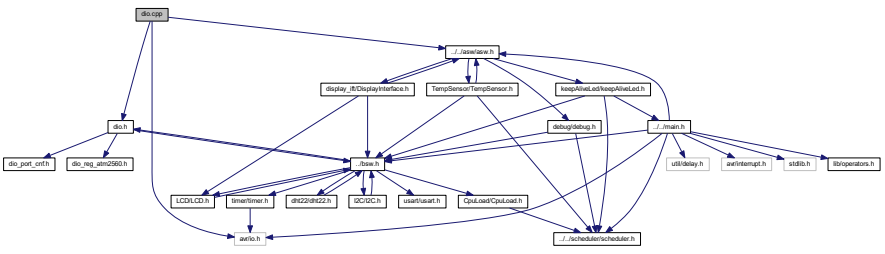
Definition at line 16 of file dht22.h.

4.11 dio.cpp File Reference

DIO library.

```
#include <avr/io.h>
#include "dio.h"
#include "../../asw/asw.h"
```

Include dependency graph for dio.cpp:



4.11.1 Detailed Description

DIO library.

Date

13 mars 2018

Author

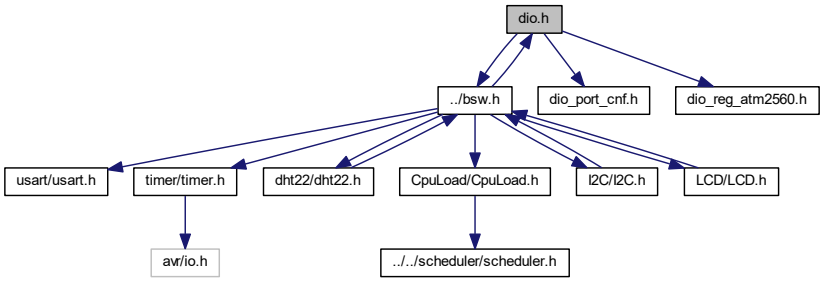
nicls67

4.12 dio.h File Reference

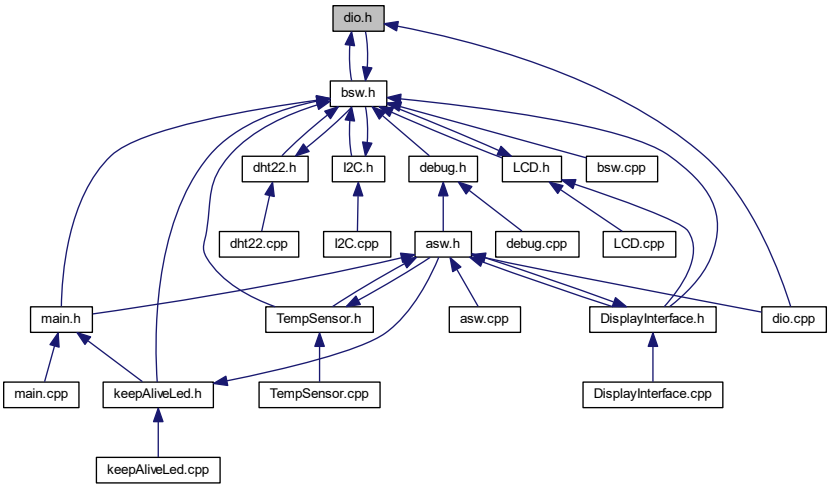
DIO library header file.

```
#include "../bsw.h"
#include "dio_port_cnf.h"
#include "dio_reg_atm2560.h"
```

Include dependency graph for dio.h:



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



Classes

- class dio

  DIO class.

Macros

- #define PORT_CNF_OUT 1
- #define PORT_CNF_IN 0
- #define ENCODE_PORT(port, pin) (uint8_t)((((uint8_t)(port & 0xF)) << 3) | (uint8_t)(pin & 0x7))
- #define DECODE_PORT(portcode) (uint8_t)((portcode >> 3) & 0xF)
- #define DECODE_PIN(portcode) (uint8_t)(portcode & 0x7)
- #define PORT_A 0
- #define PORT_B 1
- #define PORT_C 2
- #define PORT_D 3

4.12.1 Detailed Description

DIO library header file.

Date

13 mars 2018

Author

nicls67

4.12.2 Macro Definition Documentation

4.12.2.1 DECODE_PIN

```
#define DECODE_PIN(
              portcode ) (uint8_t)(portcode & 0x7)
```

Macro used to extract pin index

Definition at line 19 of file dio.h.

4.12.2.2 DECODE_PORT

```
#define DECODE_PORT(
             portcode ) (uint8_t)((portcode >> 3) & 0xF)
```

Macro used to extract port index

Definition at line 18 of file dio.h.

4.12.2.3 ENCODE_PORT

```
#define ENCODE_PORT(
             port,
             pin ) (uint8_t)((((uint8_t)(port & 0xF)) << 3) | (uint8_t)(pin & 0x7))
```

Macro used to encode port and pin indexes into one single byte

Definition at line 17 of file dio.h.

4.12.2.4 PORT_A

```
#define PORT_A 0
```

PORTA index

Definition at line 21 of file dio.h.

4.12.2.5 PORT_B

```
#define PORT_B 1
```

PORTB index

Definition at line 22 of file dio.h.

4.12.2.6 PORT_C

```
#define PORT_C 2
```

PORTC index

Definition at line 23 of file dio.h.

4.12.2.7 PORT_CNF_IN

```
#define PORT_CNF_IN 0
```

Pin is configured as input

Definition at line 15 of file dio.h.

4.12.2.8 PORT_CNF_OUT

```
#define PORT_CNF_OUT 1
```

Pin is configured as output

Definition at line 14 of file dio.h.

4.12.2.9 PORT_D

```
#define PORT_D 3
```

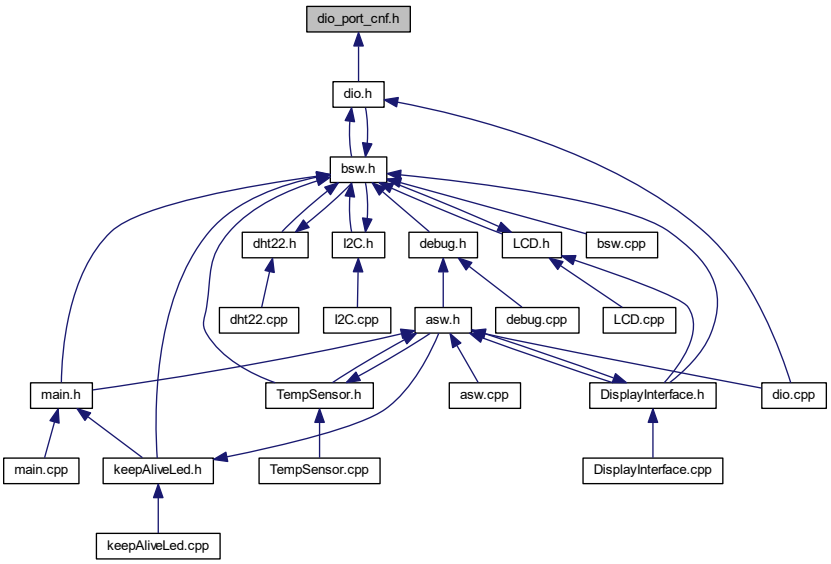
PORTD index

Definition at line 24 of file dio.h.

4.13 dio_port_cnf.h File Reference

Digital ports configuration file.

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



Macros

- #define PORTB_CNF_DDRB (uint8_t)0b11000000

  Defines the configuration of DDRB register.
- #define PORTB_CNF_PORTB (uint8_t)0b11000000

  Defines the configuration of PORTB register.

4.13.1 Detailed Description

Digital ports configuration file.

Date

19 mars 2019

Author

nicls67

4.13.2 Macro Definition Documentation

4.13.2.1 PORTB_CNF_DDRB

```
#define PORTB_CNF_DDRB (uint8_t)0b11000000
```

Defines the configuration of DDRB register.

This constant defines the direction of IO pins of PORT B. It will configure register DDRB.
PB0 : N/A
PB1 : N/A
PB2 : N/A
PB3 : N/A
PB4 : N/A
PB5 : N/A
PB6 : OUT
PB7 : OUT

Definition at line 25 of file dio_port_cnf.h.

4.13.2.2 PORTB_CNF_PORTB

```
#define PORTB_CNF_PORTB (uint8_t)0b11000000
```

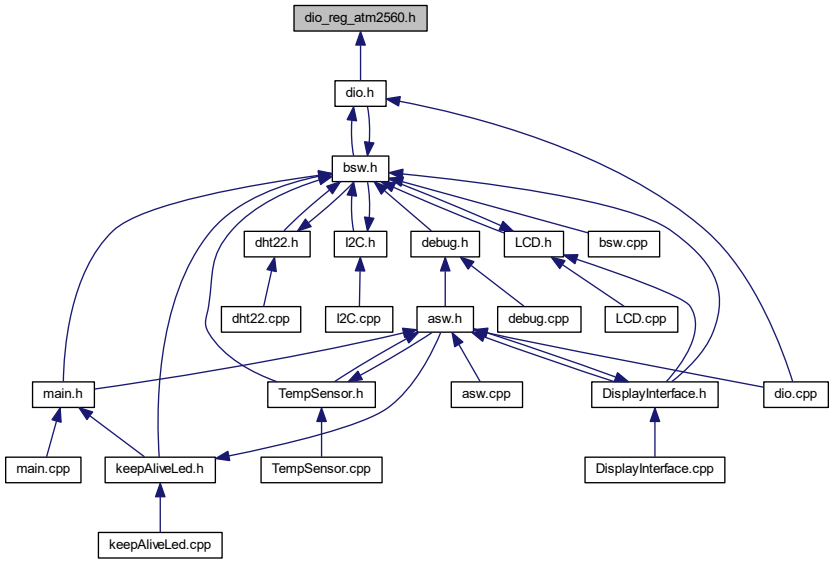
Defines the configuration of PORTB register.

This constant defines the initial value of IO pins for PORT B. It will configure register PORTB. Pins configured as input shall not be configured here.
PB0 : N/A
PB1 : N/A
PB2 : N/A
PB3 : N/A
PB4 : N/A
PB5 : N/A
PB6 : HIGH
PB7 : HIGH

Definition at line 40 of file dio_port_cnf.h.

4.14 dio_reg_atm2560.h File Reference

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



Macros

- #define PORTA_PTR (volatile uint8_t *)(0x02 + 0x20)
- #define PORTB_PTR (volatile uint8_t *)(0x05 + 0x20)
- #define PORTC_PTR (volatile uint8_t *)(0x08 + 0x20)
- #define PORTD_PTR (volatile uint8_t *)(0x0B + 0x20)
- #define PINA_PTR (volatile uint8_t *)(0x00 + 0x20)

- #define PINB_PTR (volatile uint8_t *)(0x03 + 0x20)
- #define PINC_PTR (volatile uint8_t *)(0x06 + 0x20)
- #define PIND_PTR (volatile uint8_t *)(0x09 + 0x20)
- #define DDRA_PTR (volatile uint8_t *)(0x01 + 0x20)
- #define DDRB_PTR (volatile uint8_t *)(0x04 + 0x20)
- #define DDRC_PTR (volatile uint8_t *)(0x07 + 0x20)
- #define DDRD_PTR (volatile uint8_t *)(0x0A + 0x20)

4.14.1 Macro Definition Documentation

4.14.1.1 DDRA_PTR

```
#define DDRA_PTR (volatile uint8_t *)(0x01 + 0x20)
```

Macro defining pointer to DDR A register

Definition at line 24 of file dio_reg_atm2560.h.

4.14.1.2 DDRB_PTR

```
#define DDRB_PTR (volatile uint8_t *)(0x04 + 0x20)
```

Macro defining pointer to DDR B register

Definition at line 25 of file dio_reg_atm2560.h.

4.14.1.3 DDRC_PTR

```
#define DDRC_PTR (volatile uint8_t *)(0x07 + 0x20)
```

Macro defining pointer to DDR C register

Definition at line 26 of file dio_reg_atm2560.h.

4.14.1.4 DDRD_PTR

```
#define DDRD_PTR (volatile uint8_t *)(0x0A + 0x20)
```

Macro defining pointer to DDR D register

Definition at line 27 of file dio_reg_atm2560.h.

4.14.1.5 PINA_PTR

```
#define PINA_PTR (volatile uint8_t *)(0x00 + 0x20)
```

Macro defining pointer to PIN A register

Definition at line 19 of file dio_reg_atm2560.h.

4.14.1.6 PINB_PTR

```
#define PINB_PTR (volatile uint8_t *)(0x03 + 0x20)
```

Macro defining pointer to PIN B register

Definition at line 20 of file dio_reg_atm2560.h.

4.14.1.7 PINC_PTR

```
#define PINC_PTR (volatile uint8_t *)(0x06 + 0x20)
```

Macro defining pointer to PIN C register

Definition at line 21 of file dio_reg_atm2560.h.

4.14.1.8 PIND_PTR

```
#define PIND_PTR (volatile uint8_t *)(0x09 + 0x20)
```

Macro defining pointer to PIN D register

Definition at line 22 of file dio_reg_atm2560.h.

4.14.1.9 PORTA_PTR

```
#define PORTA_PTR (volatile uint8_t *)(0x02 + 0x20)
```

Macro defining pointer to PORT A register

Definition at line 14 of file dio_reg_atm2560.h.

4.14.1.10 PORTB_PTR

```
#define PORTB_PTR (volatile uint8_t *)(0x05 + 0x20)
```

Macro defining pointer to PORT B register

Definition at line 15 of file dio_reg_atm2560.h.

4.14.1.11 PORTC_PTR

```
#define PORTC_PTR (volatile uint8_t *)(0x08 + 0x20)
```

Macro defining pointer to PORT C register

Definition at line 16 of file dio_reg_atm2560.h.

4.14.1.12 PORTD_PTR

```
#define PORTD_PTR (volatile uint8_t *)(0x0B + 0x20)
```

Macro defining pointer to PORT D register

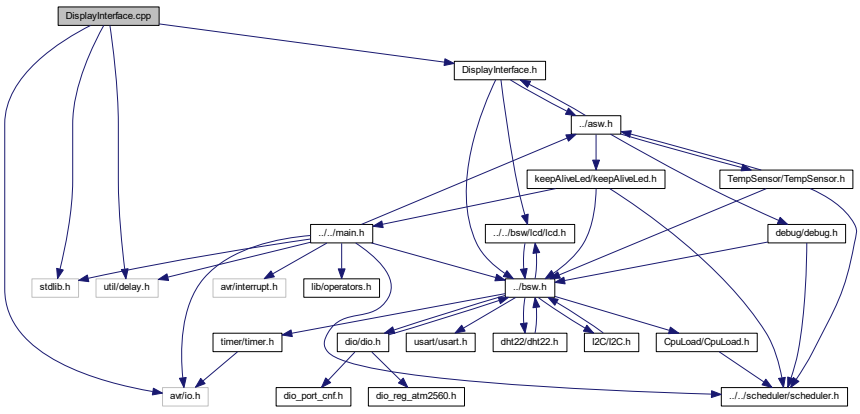
Definition at line 17 of file dio_reg_atm2560.h.

4.15 DisplayInterface.cpp File Reference

Source code file for display services.

```
#include <stdlib.h>
#include <avr/io.h>
#include <util/delay.h>
#include "DisplayInterface.h"
```

Include dependency graph for DisplayInterface.cpp:



4.15.1 Detailed Description

Source code file for display services.

Date

23 avr. 2019

Author

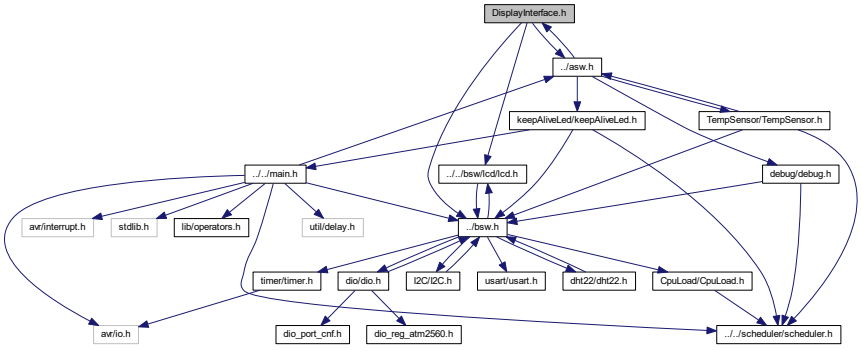
nicls67

4.16 DisplayInterface.h File Reference

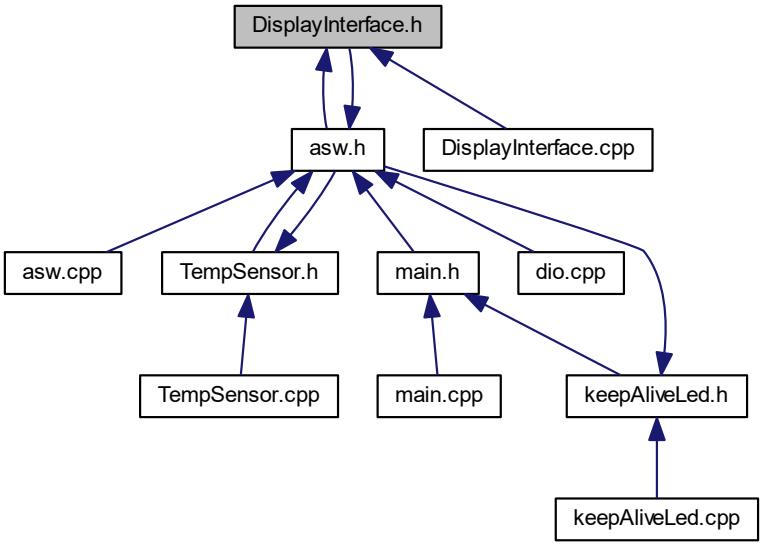
DisplayInterface class header file.

```
#include "../../bsw/lcd/lcd.h"
#include "../../bsw/bsw.h"
#include "../asw.h"
```

Include dependency graph for DisplayInterface.h:



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



Classes

- class DisplayInterface

  Display interface services class.

Enumerations

- enum T_DisplayInterface_LineDisplayMode { NORMAL, LINE_SHIFT, GO_TO_NEXT_LINE }

  Modes for line display.

Variables

- const T_LCD_conf_struct LCD_init_cnf

4.16.1 Detailed Description

DisplayInterface class header file.

Date

23 avr. 2019

Author

nicls67

4.16.2 Enumeration Type Documentation

4.16.2.1 T_DisplayInterface_LineDisplayMode

```
enum T_DisplayInterface_LineDisplayMode
```

Modes for line display.

This enumeration defines the available modes for the line display functionality :
1- Normal mode : if the string is too long to be displayed entirely, the end of the string is cut.
2- Line shift mode : the display is moving to display all the string.
3- Next line mode : the remaining characters are displayed on the next lines.

Enumerator

NORMAL	
LINE_SHIFT	
GO_TO_NEXT_LINE	

Definition at line 33 of file DisplayInterface.h.

4.16.3 Variable Documentation

4.16.3.1 LCD_init_cnf

```
const T_LCD_conf_struct LCD_init_cnf
```

Initial value:

```
= {
    LCD_CNF_BACKLIGHT_ON,
    LCD_CNF_TWO_LINE,
    LCD_CNF_FONT_5_8,
    LCD_CNF_DISPLAY_ON,
    LCD_CNF_CURSOR_OFF,
    LCD_CNF_CURSOR_BLINK_OFF,
    LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT,
    LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF
}
```

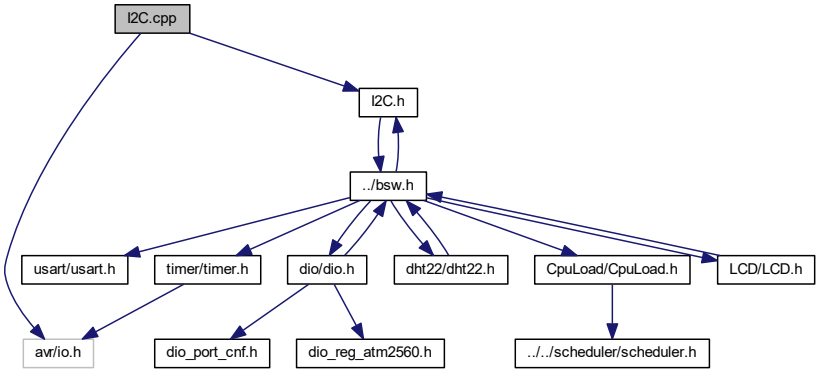
Definition at line 15 of file DisplayInterface.h.

4.17 I2C.cpp File Reference

Two-wire interface (I2C) source file.

```
#include <avr/io.h>
#include "I2C.h"
```

Include dependency graph for I2C.cpp:



4.17.1 Detailed Description

Two-wire interface (I2C) source file.

Date

19 avr. 2019

Author

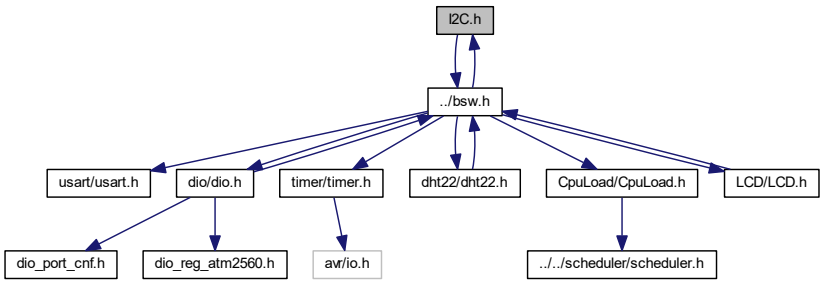
nicls67

4.18 I2C.h File Reference

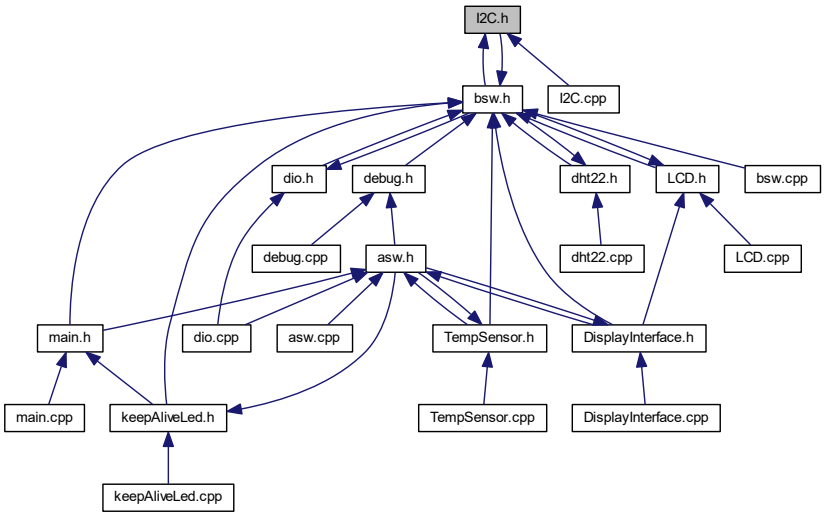
I2C class header file.

```
#include "../bsw.h"
```

Include dependency graph for I2C.h:



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



Classes

- class I2C

  Two-wire serial interface (I2C) class definition.

Macros

- #define START 0x08
- #define SLA_ACK 0x18
- #define DATA_ACK 0x28

4.18.1 Detailed Description

I2C class header file.

Date

19 avr. 2019

Author

nicls67

4.18.2 Macro Definition Documentation

4.18.2.1 DATA_ACK

```
#define DATA_ACK 0x28
```

TWSR status code : DATA has been transmitted and ACK has been received

Definition at line 15 of file I2C.h.

4.18.2.2 SLA_ACK

```
#define SLA_ACK 0x18
```

TWSR status code : SLA has been transmitted and ACK has been received

Definition at line 14 of file I2C.h.

4.18.2.3 START

```
#define START 0x08
```

TWSR status code : START condition transmitted

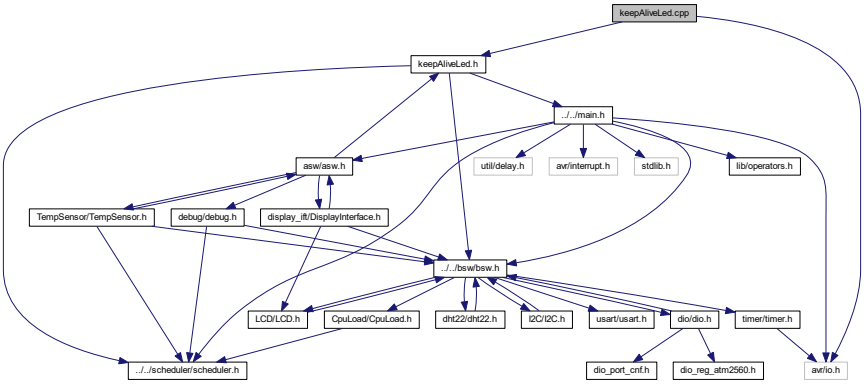
Definition at line 13 of file I2C.h.

4.19 keepAliveLed.cpp File Reference

Definition of function for class keepAliveLed.

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

Include dependency graph for keepAliveLed.cpp:



4.19.1 Detailed Description

Definition of function for class keepAliveLed.

Date

17 mars 2018

Author

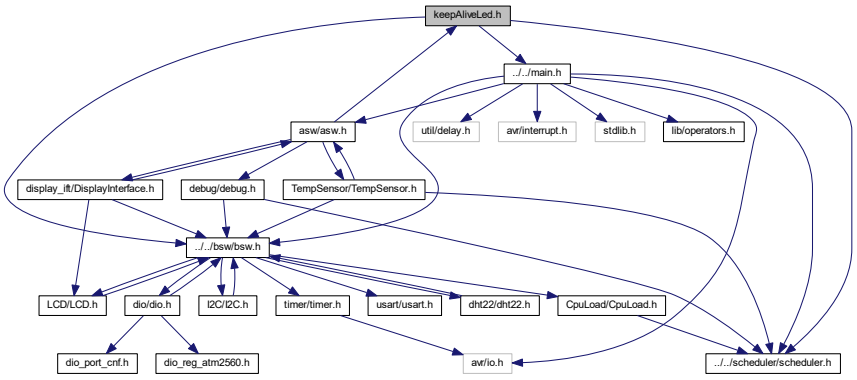
nicls67

4.20 keepAliveLed.h File Reference

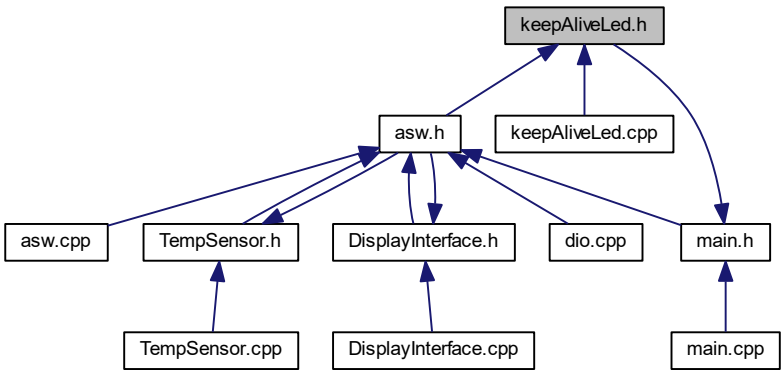
Class keepAliveLed header file.

```
#include "../../bsw/bsw.h"
#include "../../scheduler/scheduler.h"
#include "../../main.h"
```

Include dependency graph for keepAliveLed.h:



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



Classes

- class keepAliveLed

  Class for keep-alive LED blinking.

Macros

- #define PERIOD_MS_TASK_LED SW_PERIOD_MS
- #define LED_PORT ENCODE_PORT(PORT_B, 7)

4.20.1 Detailed Description

Class keepAliveLed header file.

Date

17 mars 2018

Author

nicls67

4.20.2 Macro Definition Documentation

4.20.2.1 LED_PORT

```
#define LED_PORT ENCODE_PORT(PORT_B, 7)
```

LED is connected to port PB7

Definition at line 16 of file keepAliveLed.h.

4.20.2.2 PERIOD_MS_TASK_LED

```
#define PERIOD_MS_TASK_LED SW_PERIOD_MS
```

Period for led blinking

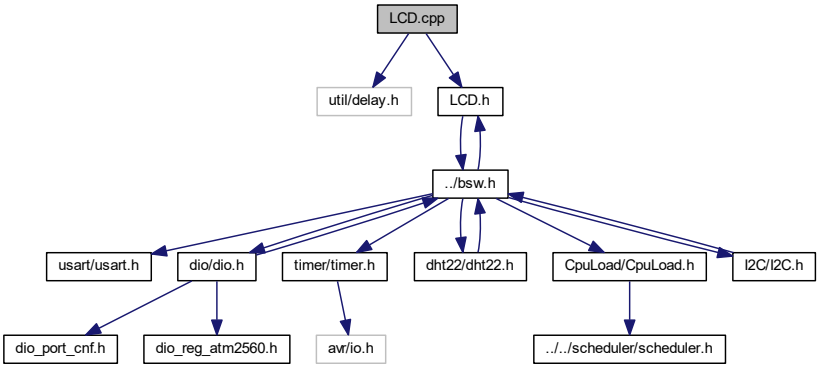
Definition at line 15 of file keepAliveLed.h.

4.21 LCD.cpp File Reference

LCD class source file.

```
#include <util/delay.h>
#include "LCD.h"
```

Include dependency graph for LCD.cpp:



4.21.1 Detailed Description

LCD class source file.

Date

20 avr. 2019

Author

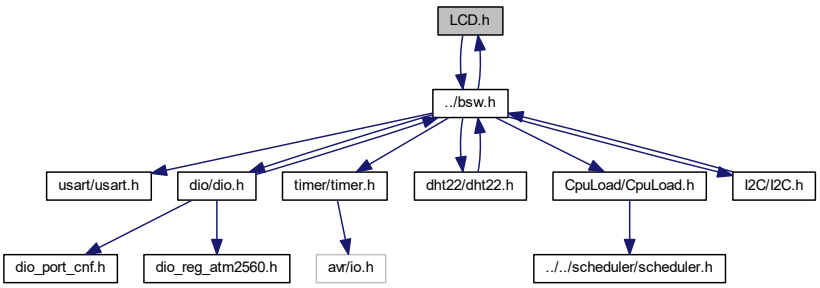
nicls67

4.22 LCD.h File Reference

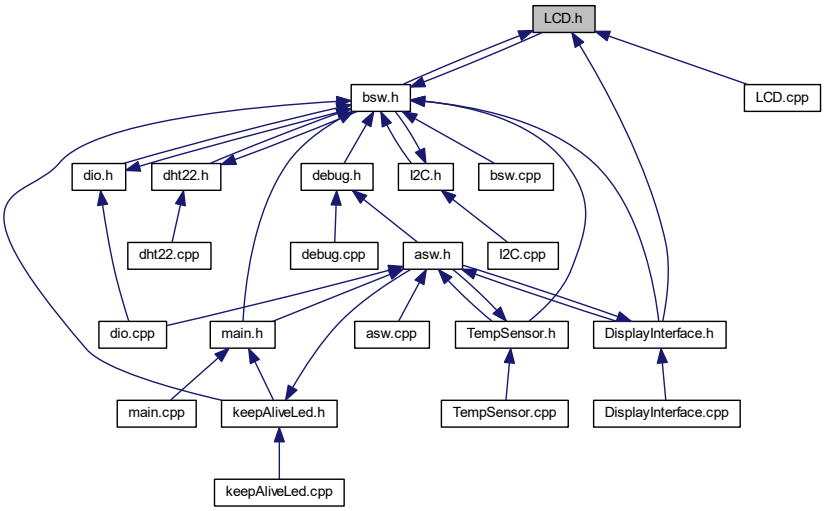
LCD class header file.

```
#include "../bsw.h"
```

Include dependency graph for LCD.h:



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



Classes

- struct T_LCD_conf_struct
- class LCD

  Class for LCD S2004A display driver.

Macros

- #define I2C_ADDR 0x27
- #define EN_PIN 2
- #define RW_PIN 1
- #define RS_PIN 0
- #define BACKLIGHT_PIN 3
- #define LCD_INST_CLR_DISPLAY_BIT 0
- #define LCD_INST_FUNCTION_SET 5
- #define LCD_INST_DISPLAY_CTRL 3
- #define LCD_INST_ENTRY_MODE_SET 2
- #define LCD_INST_SET_DDRAM_ADDR 7
- #define LCD_FCT_SET_FIELD_DL 4
- #define LCD_FCT_SET_FIELD_N 3
- #define LCD_FCT_SET_FIELD_F 2
- #define LCD_DISPLAY_CTRL_FIELD_D 2
- #define LCD_DISPLAY_CTRL_FIELD_C 1
- #define LCD_DISPLAY_CTRL_FIELD_B 0
- #define LCD_CNF_SHIFT_ID 1
- #define LCD_CNF_SHIFT_SH 0
- #define LCD_CNF_ONE_LINE 0
- #define LCD_CNF_TWO_LINE 1
- #define LCD_CNF_FONT_5_8 0
- #define LCD_CNF_FONT_5_11 1

- #define LCD_CNF_DISPLAY_ON 1
- #define LCD_CNF_DISPLAY_OFF 0
- #define LCD_CNF_CURSOR_ON 1
- #define LCD_CNF_CURSOR_OFF 0
- #define LCD_CNF_CURSOR_BLINK_ON 1
- #define LCD_CNF_CURSOR_BLINK_OFF 0
- #define LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT 1
- #define LCD_CNF_ENTRY_MODE_DIRECTION_LEFT 0
- #define LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON 1
- #define LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF 0
- #define LCD_CNF_BACKLIGHT_ON 1
- #define LCD_CNF_BACKLIGHT_OFF 0
- #define LCD_RAM_1_LINE_MIN 0
- #define LCD_RAM_1_LINE_MAX 0x4F
- #define LCD_RAM_2_LINES_MIN_1 0
- #define LCD_RAM_2_LINES_MAX_1 0x27
- #define LCD_RAM_2_LINES_MIN_2 0x40
- #define LCD_RAM_2_LINES_MAX_2 0x67
- #define LCD_WAIT_CLR_RETURN 1600
- #define LCD_WAIT_OTHER_MODES 40
- #define LCD_SIZE_NB_CHAR_PER_LINE 20
- #define LCD_SIZE_NB_LINES 4

Enumerations

- enum T_LCD_command { LCD_CMD_FUNCTION_SET, LCD_CMD_CLEAR_DISPLAY, LCD_CMD_DISPLAY_CTRL, LCD_CMD_ENTRY_MODE_SET, LCD_CMD_SET_DDRAM_ADDR }

  LCD commands enumeration.
- enum T_LCD_config_mode { LCD_MODE_INSTRUCTION = 0, LCD_MODE_DATA = 1 }

  LCD modes enumeration.
- enum T_LCD_ram_area { LCD_DATA_DDRAM, LCD_DATA_CGRAM }

  Screen RAM definition.

4.22.1 Detailed Description

LCD class header file.

Date

20 avr. 2019

Author

nicls67

4.22.2 Macro Definition Documentation

4.22.2.1 BACKLIGHT_PIN

```
#define BACKLIGHT_PIN 3
```

Backlight pin is on P3

Definition at line 19 of file LCD.h.

4.22.2.2 EN_PIN

```
#define EN_PIN 2
```

EN bit is on P2

Definition at line 16 of file LCD.h.

4.22.2.3 I2C_ADDR

```
#define I2C_ADDR 0x27
```

I2C address of LCD display

Definition at line 13 of file LCD.h.

4.22.2.4 LCD_CNF_BACKLIGHT_OFF

```
#define LCD_CNF_BACKLIGHT_OFF 0
```

Backlight is disabled

Definition at line 72 of file LCD.h.

4.22.2.5 LCD_CNF_BACKLIGHT_ON

```
#define LCD_CNF_BACKLIGHT_ON 1
```

Backlight is enabled

Definition at line 71 of file LCD.h.

4.22.2.6 LCD_CNF_CURSOR_BLINK_OFF

```
#define LCD_CNF_CURSOR_BLINK_OFF 0
```

Cursor blinking is off, bit is set to 0

Definition at line 60 of file LCD.h.

4.22.2.7 LCD_CNF_CURSOR_BLINK_ON

```
#define LCD_CNF_CURSOR_BLINK_ON 1
```

Cursor blinking is on, bit is set to 1

Definition at line 59 of file LCD.h.

4.22.2.8 LCD_CNF_CURSOR_OFF

```
#define LCD_CNF_CURSOR_OFF 0
```

Cursor is off, bit is set to 0

Definition at line 56 of file LCD.h.

4.22.2.9 LCD_CNF_CURSOR_ON

```
#define LCD_CNF_CURSOR_ON 1
```

Cursor is on, bit is set to 1

Definition at line 55 of file LCD.h.

4.22.2.10 LCD_CNF_DISPLAY_OFF

```
#define LCD_CNF_DISPLAY_OFF 0
```

Display is off, bit is set to 0

Definition at line 52 of file LCD.h.

4.22.2.11 LCD_CNF_DISPLAY_ON

```
#define LCD_CNF_DISPLAY_ON 1
```

Display is on, bit is set to 1

Definition at line 51 of file LCD.h.

4.22.2.12 LCD_CNF_ENTRY_MODE_DIRECTION_LEFT

```
#define LCD_CNF_ENTRY_MODE_DIRECTION_LEFT 0
```

Direction of shift is left, bit is set to 0

Definition at line 64 of file LCD.h.

4.22.2.13 LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT

```
#define LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT 1
```

Direction of shift is right, bit is set to 1

Definition at line 63 of file LCD.h.

4.22.2.14 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF

```
#define LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF 0
```

Display shift is not performed, bit is set to 0

Definition at line 68 of file LCD.h.

4.22.2.15 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON

```
#define LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON 1
```

Display shift is performed, bit is set to 1

Definition at line 67 of file LCD.h.

4.22.2.16 LCD_CNF_FONT_5_11

```
#define LCD_CNF_FONT_5_11 1
```

Two-line configuration, bit is set to 1

Definition at line 48 of file LCD.h.

4.22.2.17 LCD_CNF_FONT_5_8

```
#define LCD_CNF_FONT_5_8 0
```

One-line configuration, bit is set to 0

Definition at line 47 of file LCD.h.

4.22.2.18 LCD_CNF_ONE_LINE

```
#define LCD_CNF_ONE_LINE 0
```

One-line configuration, bit is set to 0

Definition at line 43 of file LCD.h.

4.22.2.19 LCD_CNF_SHIFT_ID

```
#define LCD_CNF_SHIFT_ID 1
```

Field ID (increment or decrement) of command "entry mode set" is on bit DB1

Definition at line 39 of file LCD.h.

4.22.2.20 LCD_CNF_SHIFT_SH

```
#define LCD_CNF_SHIFT_SH 0
```

Field SH (shift of display) of command "entry mode set" is on bit DB1

Definition at line 40 of file LCD.h.

4.22.2.21 LCD_CNF_TWO_LINE

```
#define LCD_CNF_TWO_LINE 1
```

Two-line configuration, bit is set to 1

Definition at line 44 of file LCD.h.

4.22.2.22 LCD_DISPLAY_CTRL_FIELD_B

```
#define LCD_DISPLAY_CTRL_FIELD_B 0
```

Field B (cursor blink) of command "display control" is on bit DB0

Definition at line 36 of file LCD.h.

4.22.2.23 LCD_DISPLAY_CTRL_FIELD_C

```
#define LCD_DISPLAY_CTRL_FIELD_C 1
```

Field C (cursor on/off) of command "display control" is on bit DB1

Definition at line 35 of file LCD.h.

4.22.2.24 LCD_DISPLAY_CTRL_FIELD_D

```
#define LCD_DISPLAY_CTRL_FIELD_D 2
```

Field D (display on/off) of command "display control" is on bit DB2

Definition at line 34 of file LCD.h.

4.22.2.25 LCD_FCT_SET_FIELD_DL

```
#define LCD_FCT_SET_FIELD_DL 4
```

Field DL (data length) of command "function set" is on bit DB4

Definition at line 29 of file LCD.h.

4.22.2.26 LCD_FCT_SET_FIELD_F

```
#define LCD_FCT_SET_FIELD_F 2
```

Field F (font type) of command "function set" is on bit DB2

Definition at line 31 of file LCD.h.

4.22.2.27 LCD_FCT_SET_FIELD_N

```
#define LCD_FCT_SET_FIELD_N 3
```

Field N (number of lines) of command "function set" is on bit DB3

Definition at line 30 of file LCD.h.

4.22.2.28 LCD_INST_CLR_DISPLAY_BIT

```
#define LCD_INST_CLR_DISPLAY_BIT 0
```

Instruction bit for "clear display" is DB0

Definition at line 22 of file LCD.h.

4.22.2.29 LCD_INST_DISPLAY_CTRL

```
#define LCD_INST_DISPLAY_CTRL 3
```

Instruction bit for "display control" is DB3

Definition at line 24 of file LCD.h.

4.22.2.30 LCD_INST_ENTRY_MODE_SET

```
#define LCD_INST_ENTRY_MODE_SET 2
```

Instruction bit for "entry mode" is DB2

Definition at line 25 of file LCD.h.

4.22.2.31 LCD_INST_FUNCTION_SET

```
#define LCD_INST_FUNCTION_SET 5
```

Instruction bit for "function set" is DB5

Definition at line 23 of file LCD.h.

4.22.2.32 LCD_INST_SET_DDRAM_ADDR

```
#define LCD_INST_SET_DDRAM_ADDR 7
```

Instruction bit for "set DDRAM address" is DB7

Definition at line 26 of file LCD.h.

4.22.2.33 LCD_RAM_1_LINE_MAX

```
#define LCD_RAM_1_LINE_MAX 0x4F
```

Maximum address value in 1-line mode

Definition at line 76 of file LCD.h.

4.22.2.34 LCD_RAM_1_LINE_MIN

```
#define LCD_RAM_1_LINE_MIN 0
```

Minimum address value in 1-line mode

Definition at line 75 of file LCD.h.

4.22.2.35 LCD_RAM_2_LINES_MAX_1

```
#define LCD_RAM_2_LINES_MAX_1 0x27
```

Maximum address value in 2-lines mode for line 1

Definition at line 78 of file LCD.h.

4.22.2.36 LCD_RAM_2_LINES_MAX_2

```
#define LCD_RAM_2_LINES_MAX_2 0x67
```

Maximum address value in 2-lines mode for line 2

Definition at line 80 of file LCD.h.

4.22.2.37 LCD_RAM_2_LINES_MIN_1

```
#define LCD_RAM_2_LINES_MIN_1 0
```

Minimum address value in 2-lines mode for line 1

Definition at line 77 of file LCD.h.

4.22.2.38 LCD_RAM_2_LINES_MIN_2

```
#define LCD_RAM_2_LINES_MIN_2 0x40
```

Minimum address value in 2-lines mode for line 2

Definition at line 79 of file LCD.h.

4.22.2.39 LCD_SIZE_NB_CHAR_PER_LINE

```
#define LCD_SIZE_NB_CHAR_PER_LINE 20
```

LCD screen has 20 characters per line

Definition at line 87 of file LCD.h.

4.22.2.40 LCD_SIZE_NB_LINES

```
#define LCD_SIZE_NB_LINES 4
```

LCD screen has 4 lines

Definition at line 88 of file LCD.h.

4.22.2.41 LCD_WAIT_CLR_RETURN

```
#define LCD_WAIT_CLR_RETURN 1600
```

Waiting time after clear display and return home operations is at least 1520 us

Definition at line 83 of file LCD.h.

4.22.2.42 LCD_WAIT_OTHER_MODES

```
#define LCD_WAIT_OTHER_MODES 40
```

Waiting time after all other modes is at least 38 us

Definition at line 84 of file LCD.h.

4.22.2.43 RS_PIN

```
#define RS_PIN 0
```

RS pin is on P0

Definition at line 18 of file LCD.h.

4.22.2.44 RW_PIN

```
#define RW_PIN 1
```

RW pin is on P1

Definition at line 17 of file LCD.h.

4.22.3 Enumeration Type Documentation

4.22.3.1 T_LCD_command

```
enum T_LCD_command
```

LCD commands enumeration.

This enumeration defines all command modes available for LCD configuration

Enumerator

LCD_CMD_FUNCTION_SET	
LCD_CMD_CLEAR_DISPLAY	
LCD_CMD_DISPLAY_CTRL	
LCD_CMD_ENTRY_MODE_SET	
LCD_CMD_SET_DDRAM_ADDR	

Definition at line 95 of file LCD.h.

4.22.3.2 T_LCD_config_mode

`enum T_LCD_config_mode`

LCD modes enumeration.

This enumeration defines the possible modes for communication with LCD. Two modes are possible, DATA for writing data in RAM and INSTRUCTION for configuring the display

Enumerator

LCD_MODE_INSTRUCTION	
LCD_MODE_DATA	

Definition at line 109 of file LCD.h.

4.22.3.3 T_LCD_ram_area

`enum T_LCD_ram_area`

Screen RAM definition.

This enumeration defines the RAM areas of the LCD screen : DDRAM for display, CGRAM for characters generation

Enumerator

LCD_DATA_DDRAM	
LCD_DATA_CGRAM	

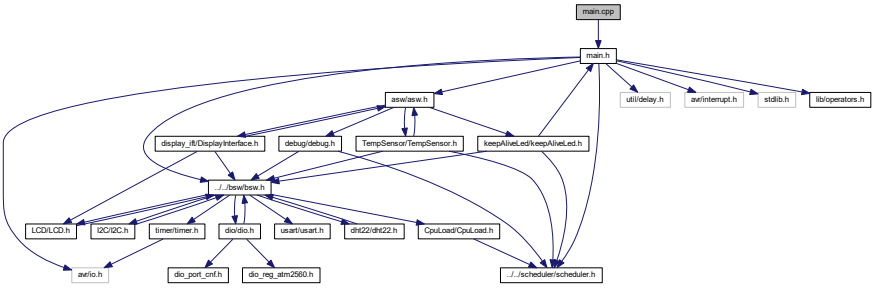
Definition at line 120 of file LCD.h.

4.23 main.cpp File Reference

Background task file.

```
#include "main.h"
```

Include dependency graph for main.cpp:



Functions

- ISR (TIMER1_COMPA_vect)

  Main software interrupt.

- ISR (USART0_RX_vect)

  USART Rx Complete interrupt.

- int main (void)

  Background task of program.

4.23.1 Detailed Description

Background task file.

Date

12 mars 2018

Author

nicls67

4.23.2 Function Documentation

4.23.2.1 ISR() [1/2]

```
ISR (
            TIMER1_COMPA_vect  )
```

Main software interrupt.

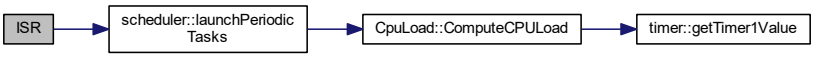
This function handles the interrupt raised by Timer #1. It wakes up the software every 500 ms to perform applications.

Returns

Nothing

Definition at line 19 of file main.cpp.

Here is the call graph for this function:



4.23.2.2 ISR() [2/2]

```
ISR (
            USART0_RX_vect  )
```

USART Rx Complete interrupt.

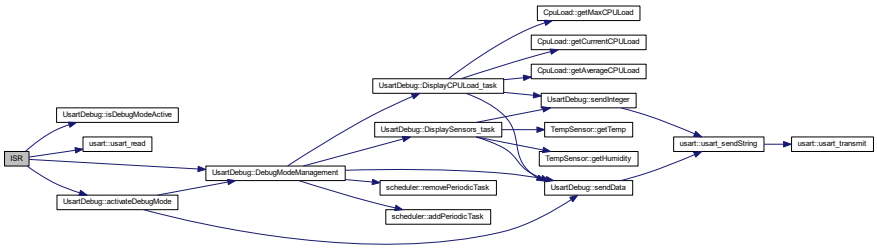
This function handles the interrupt raised when a frame has been received by USART. If debug mode mode is active, it calls debug mode management function. If inactive, it calls debug mode activation function if the received character is 'a'

Returns

Nothing

Definition at line 31 of file main.cpp.

Here is the call graph for this function:



4.23.2.3 main()

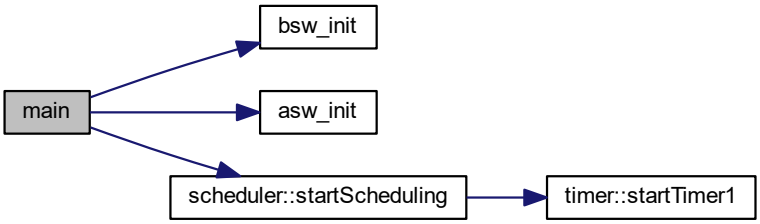
```
int main (
            void  )
```

Background task of program.

This function initializes all the software and then goes into an infinite loop. Periodic interrupt will wake up the software to perform application

Definition at line 51 of file main.cpp.

Here is the call graph for this function:

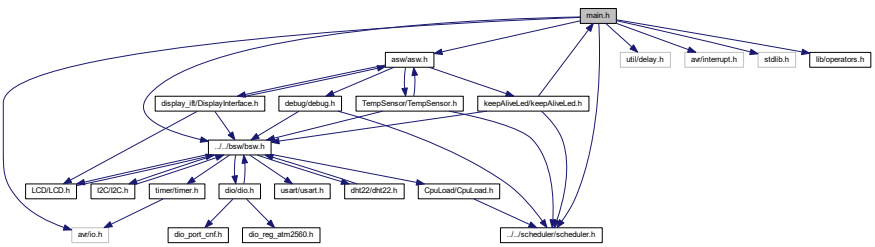


4.24 main.h File Reference

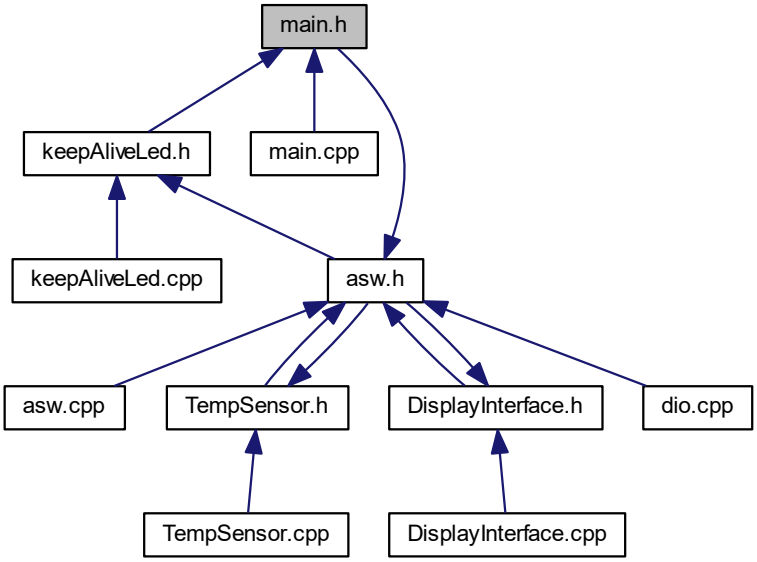
Background task header file.

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include "lib/operators.h"
#include "asw/asw.h"
#include "bsw/bsw.h"
#include "scheduler/scheduler.h"
```

Include dependency graph for main.h:



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



4.24.1 Detailed Description

Background task header file.

Date

17 mars 2018

Author

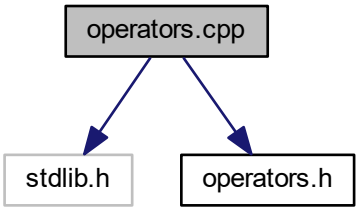
nicls67

4.25 operators.cpp File Reference

c++ operators definitions

```
#include <stdlib.h>
#include "operators.h"
```

Include dependency graph for operators.cpp:



Functions

- void * operator new (size_t a_size)

  Operator new.
- void operator delete (void *ptr)

  Operator delete.

4.25.1 Detailed Description

c++ operators definitions

Date

14 mars 2018

Author

nicls67

4.25.2 Function Documentation

4.25.2.1 operator delete()

```
void operator delete (
            void * ptr )
```

Operator delete.

Equivalent to free function in C Free the memory zone at address ptr

Parameters

in	*ptr*	Pointer to the start of memory zone to free

Returns

Nothing

Definition at line 18 of file operators.cpp.

4.25.2.2 operator new()

```
void* operator new (
            size_t a_size )
```

Operator new.

Equivalent to malloc function in C Allocates a memory zone of size a_size

Parameters

in	*a_size*	memory size to allocate

Returns

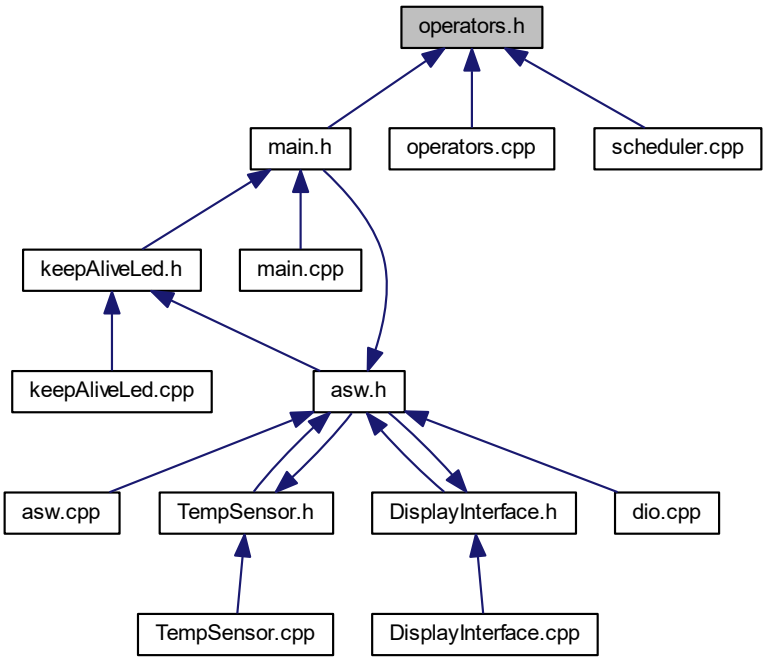
Pointer to the start of allocated memory zone

Definition at line 13 of file operators.cpp.

4.26 operators.h File Reference

c++ operators definitions header file

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



Functions

- void * operator new (size_t a_size)
  Operator new.
- void operator delete (void *ptr)
  Operator delete.

4.26.1 Detailed Description

c++ operators definitions header file

Date

14 mars 2018

Author

nicls67

4.26.2 Function Documentation

4.26.2.1 operator delete()

```
void operator delete (
            void * ptr )
```

Operator delete.

Equivalent to free function in C Free the memory zone at address ptr

Parameters

in	*ptr*	Pointer to the start of memory zone to free

Returns

Nothing

Definition at line 18 of file operators.cpp.

4.26.2.2 operator new()

```
void* operator new (
            size_t a_size )
```

Operator new.

Equivalent to malloc function in C Allocates a memory zone of size a_size

Parameters

in	*a_size*	memory size to allocate

Returns

Pointer to the start of allocated memory zone

Definition at line 13 of file operators.cpp.

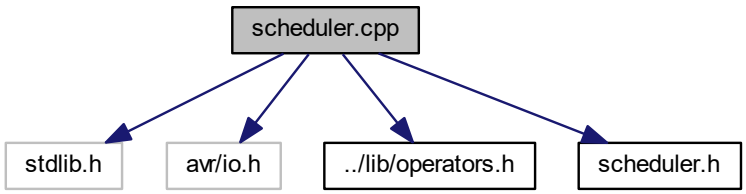
4.27 scheduler.cpp File Reference

Defines scheduler class.

```
#include <stdlib.h>
#include <avr/io.h>
#include "../lib/operators.h"
```

```
#include "scheduler.h"
```

Include dependency graph for scheduler.cpp:



Variables

- scheduler * p_scheduler

4.27.1 Detailed Description

Defines scheduler class.

Date

16 mars 2018

Author

nicls67

4.27.2 Variable Documentation

4.27.2.1 p_scheduler

```
scheduler* p_scheduler
```

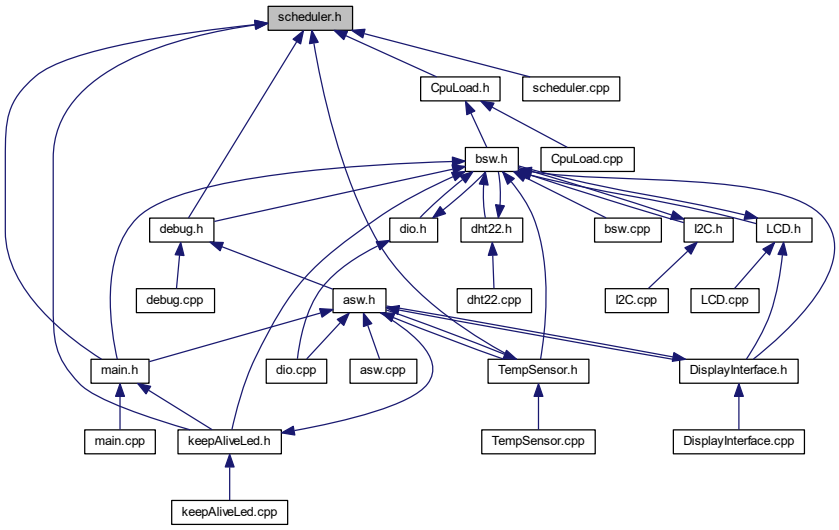
Pointer to scheduler object

Definition at line 17 of file scheduler.cpp.

4.28 scheduler.h File Reference

Scheduler class header file.

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



Classes

- class scheduler

  Scheduler class.
- struct scheduler::Task_t

  Type defining a task structure.
- struct scheduler::Task_cnf_struct_t

  Task configuration structure.

Macros

- #define SW_PERIOD_MS 500
- #define PRESCALER_PERIODIC_TIMER 256
- #define TIMER_CTC_VALUE ((F_CPU/PRESCALER_PERIODIC_TIMER)/(1000/SW_PERIOD_MS))

Typedefs

- typedef void(∗ TaskPtr_t) (void)

  Type defining a pointer to function.

Variables

- scheduler ∗ p_scheduler

4.28.1 Detailed Description

Scheduler class header file.

Date

16 mars 2018

Author

nicls67

4.28.2 Macro Definition Documentation

4.28.2.1 PRESCALER_PERIODIC_TIMER

```
#define PRESCALER_PERIODIC_TIMER 256
```

Value of prescaler to use for periodic timer

Definition at line 15 of file scheduler.h.

4.28.2.2 SW_PERIOD_MS

```
#define SW_PERIOD_MS 500
```

Software period, used to define periodic timer interrupt

Definition at line 14 of file scheduler.h.

4.28.2.3 TIMER_CTC_VALUE

```
#define TIMER_CTC_VALUE ((F_CPU/PRESCALER_PERIODIC_TIMER)/(1000/SW_PERIOD_MS))
```

Compare value for periodic timer

Definition at line 16 of file scheduler.h.

4.28.3 Typedef Documentation

4.28.3.1 TaskPtr_t

```
typedef void(* TaskPtr_t) (void)
```

Type defining a pointer to function.

Definition at line 21 of file scheduler.h.

4.28.4 Variable Documentation

4.28.4.1 p_scheduler

```
scheduler* p_scheduler
```

Pointer to scheduler object

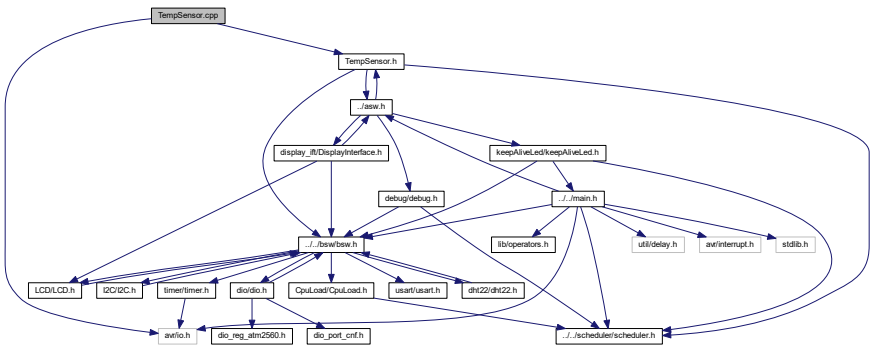
Definition at line 17 of file scheduler.cpp.

4.29 TempSensor.cpp File Reference

Defines function of class TempSensor.

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

Include dependency graph for TempSensor.cpp:



Macros

- #define PIT_BEFORE_INVALID 60

4.29.1 Detailed Description

Defines function of class TempSensor.

Date

23 mars 2018

Author

nicls67

4.29.2 Macro Definition Documentation

4.29.2.1 PIT_BEFORE_INVALID

```
#define PIT_BEFORE_INVALID 60
```

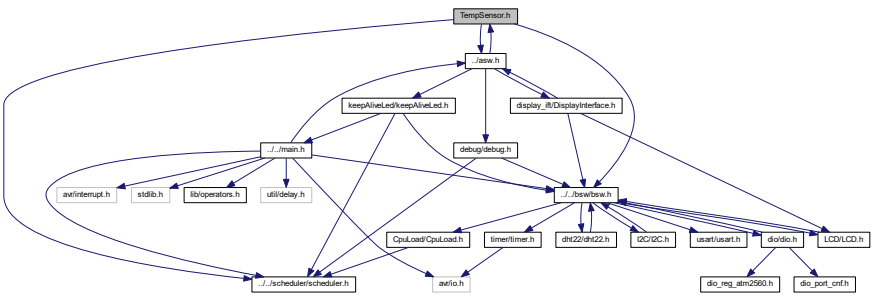
Definition at line 14 of file TempSensor.cpp.

4.30 TempSensor.h File Reference

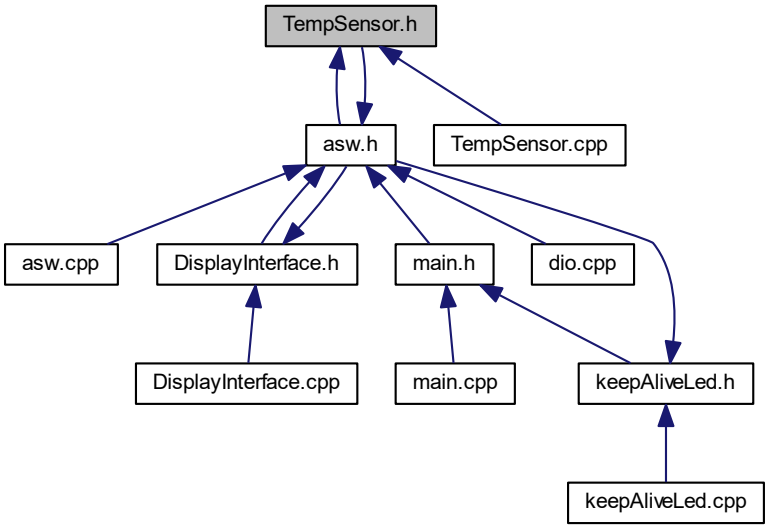
Class TempSensor header file.

```
#include "../../scheduler/scheduler.h"
#include "../../bsw/bsw.h"
#include "../asw.h"
```

Include dependency graph for TempSensor.h:



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



Classes

- class TempSensor

  Class for temperature sensor.

Macros

- #define PERIOD_MS_TASK_TEMP_SENSOR 5000

4.30.1 Detailed Description

Class TempSensor header file.

Date

23 mars 2018

Author

nicls67

4.30.2 Macro Definition Documentation

4.30.2.1 PERIOD_MS_TASK_TEMP_SENSOR

```
#define PERIOD_MS_TASK_TEMP_SENSOR 5000
```

Period for reading temperature data

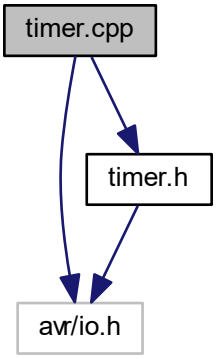
Definition at line 13 of file TempSensor.h.

4.31 timer.cpp File Reference

Defines function for class timer.

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

Include dependency graph for timer.cpp:



4.31.1 Detailed Description

Defines function for class timer.

Date

15 mars 2018

Author

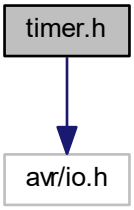
nicls67

4.32 timer.h File Reference

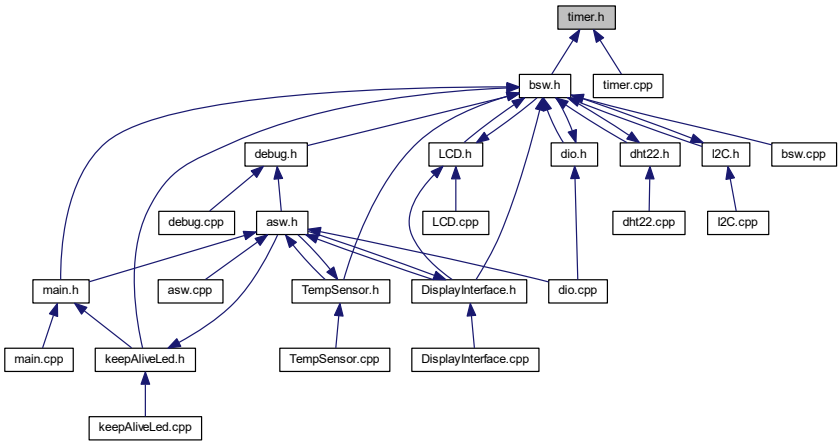
Timer class header file.

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

Include dependency graph for timer.h:



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



Classes

- class timer

  Class defining a timer.

4.32.1 Detailed Description

Timer class header file.

Date

15 mars 2018

Author

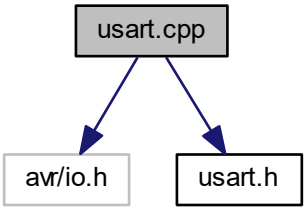
nicls67

4.33 usart.cpp File Reference

BSW library for USART.

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

Include dependency graph for usart.cpp:



4.33.1 Detailed Description

BSW library for USART.

Date

13 mars 2018

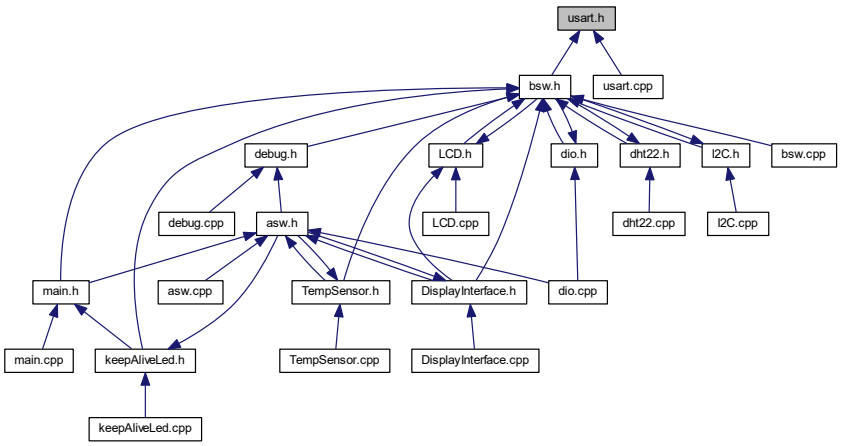
Author

nicls67

4.34 usart.h File Reference

Header file for USART library.

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



Classes

- class usart

  USART serial bus class.

4.34.1 Detailed Description

Header file for USART library.

Date

13 mars 2018

Author

nicls67

Index