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 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.

3.1.1 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.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 dht22()

`dht22::dht22 ( )`

dht22 class constructor

Initializes the class dht22

Returns

Nothing

Definition at line 22 of file dht22.cpp.

3.1.3 Member Function Documentation

3.1.3.1 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

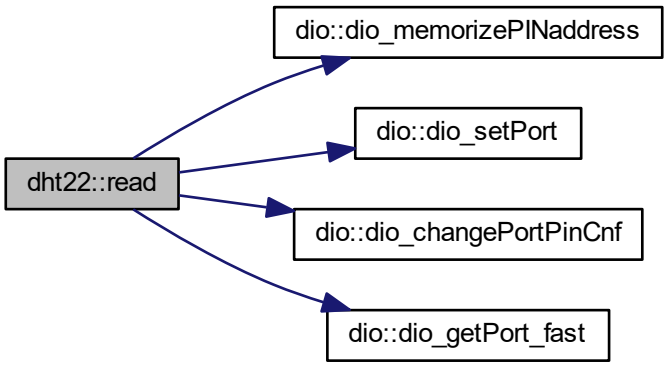
out	*raw_humidity*	Raw humidity value received from sensor
out	*raw_temperature*	Raw temperature value received from sensor

Returns

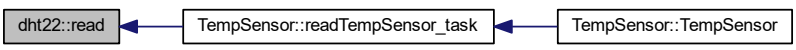
Validity of the read value

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:

- work/bsw/dht22/dht22.h
- work/bsw/dht22/dht22.cpp

3.2 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.

3.2.1 Detailed Description

DIO class.

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

Definition at line 31 of file dio.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 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.

3.2.3 Member Function Documentation

3.2.3.1 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

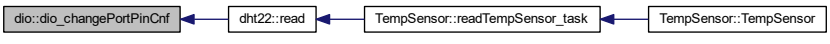
in	*portcode*	Encoded pin and register index
in	*cnf*	Requested configuration for the selected pin PORT_CNF_OUT (1) : pin configured as output PORT_CNF_IN (0) : pin configured as input

Returns

Nothing

Definition at line 149 of file dio.cpp.

Here is the caller graph for this function:



3.2.3.2 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.

3.2.3.3 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:

3.2.3.4 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 caller graph for this function:

3.2.3.5 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 caller graph for this function:

3.2.3.6 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 caller graph for this function:

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

- work/bsw/dio/dio.h
- work/bsw/dio/dio.cpp

3.3 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.

3.3.1 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.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 keepAliveLed()

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

Class constructor.

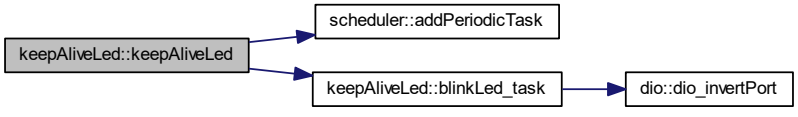
This function initializes the class keppAliveLed

Returns

Nothing

Definition at line 15 of file keepAliveLed.cpp.

Here is the call graph for this function:



3.3.3 Member Function Documentation

3.3.3.1 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:

- work/asw/keepAliveLed/keepAliveLed.h
- work/asw/keepAliveLed/keepAliveLed.cpp

3.4 scheduler Class Reference

Scheduler class.

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

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.

3.4.1 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 32 of file scheduler.h.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 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:

3.4.3 Member Function Documentation

3.4.3.1 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

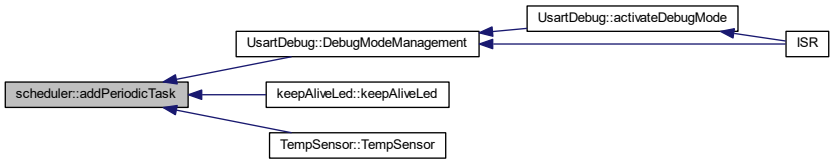
in	*task_ptr*	Pointer to the task which will be added
in	*a_period*	Period of the new task

Returns

Nothing

Definition at line 63 of file scheduler.cpp.

Here is the caller graph for this function:



3.4.3.2 getPitNumber()

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

Get function for PIT number.

This function returns the PIT number

Returns

PIT number

Definition at line 93 of file scheduler.cpp.

Here is the caller graph for this function:

3.4.3.3 launchPeriodicTasks()

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

Main scheduler function.

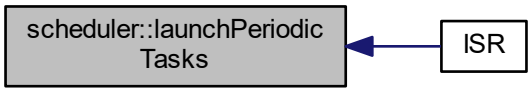
This function launches the scheduled tasks according to current software time and task configuration

Returns

Nothing

Definition at line 32 of file scheduler.cpp.

Here is the caller graph for this function:



3.4.3.4 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 99 of file scheduler.cpp.

Here is the caller graph for this function:

3.4.3.5 startScheduling()

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

Starts the tasks scheduling.

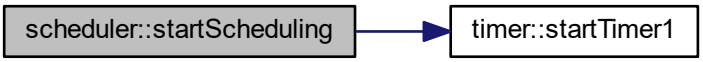
This function starts the timer which will trigger an interrupt every software period. When the interrupt is raised the scheduler will launch applications

Returns

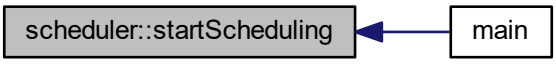
Nothing

Definition at line 57 of file scheduler.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:

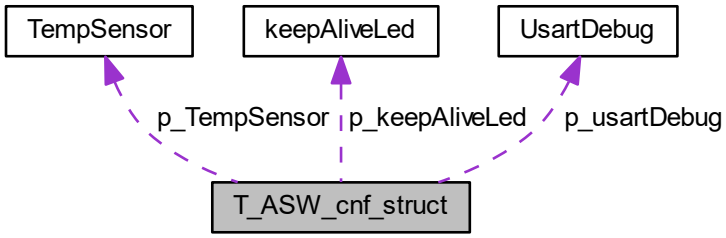
- work/scheduler/scheduler.h
- work/scheduler/scheduler.cpp

3.5 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

3.5.1 Detailed Description

ASW configuration structure.

This structure contains all pointers to instanced applicative objects

Definition at line 23 of file asw.h.

3.5.2 Member Data Documentation

3.5.2.1 p_keepAliveLed

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

Pointer to keepAliveLed object

Definition at line 26 of file asw.h.

3.5.2.2 p_TempSensor

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

Pointer to TempSensor object

Definition at line 27 of file asw.h.

3.5.2.3 p_usartDebug

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

Pointer to usart debug object

Definition at line 25 of file asw.h.

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

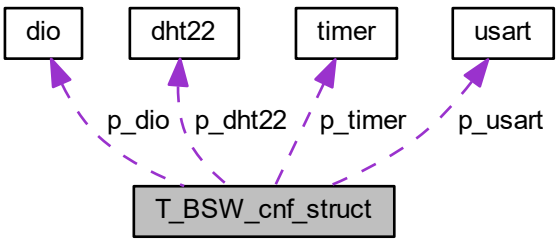
- work/asw/asw.h

3.6 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

3.6.1 Detailed Description

BSW configuration structure.

This structure contains all pointers to instanced drivers objects

Definition at line 29 of file bsw.h.

3.6.2 Member Data Documentation

3.6.2.1 p_dht22

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

Pointer to dht22 driver object

Definition at line 34 of file bsw.h.

3.6.2.2 p_dio

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

Pointer to dio driver object

Definition at line 32 of file bsw.h.

3.6.2.3 p_timer

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

Pointer to timer driver object

Definition at line 33 of file bsw.h.

3.6.2.4 p_usart

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

Pointer to usart driver object

Definition at line 31 of file bsw.h.

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

- work/bsw/bsw.h

3.7 TempSensor Class Reference

```
#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.

3.7.1 Detailed Description

Definition at line 15 of file TempSensor.h.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 TempSensor()

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

Class constructor.

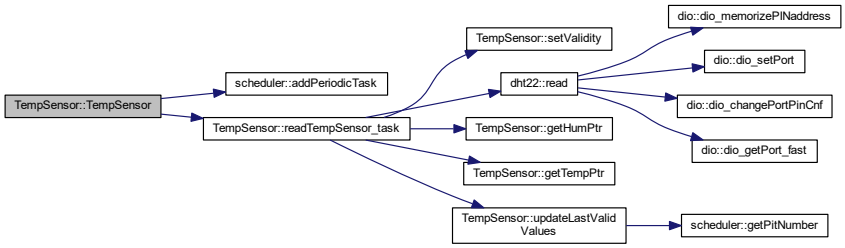
This function initializes all data of the class TempSensor

Returns

Nothing

Definition at line 16 of file TempSensor.cpp.

Here is the call graph for this function:



3.7.3 Member Function Documentation

3.7.3.1 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

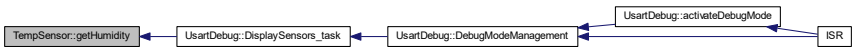
out	*hum*	Humidity value

Returns

Validity of humidity

Definition at line 66 of file TempSensor.cpp.

Here is the caller graph for this function:



3.7.3.2 getHumPtr()

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

Get pointer to data raw_humidity.

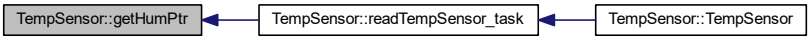
This function returns a pointer to the class member raw_humidity

Returns

Pointer to raw_humidity

Definition at line 41 of file TempSensor.cpp.

Here is the caller graph for this function:



3.7.3.3 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:

3.7.3.4 getTempPtr()

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

Get pointer to data raw_temperature.

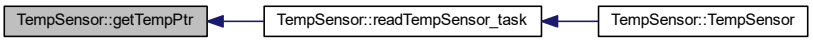
This function returns a pointer to the class member raw_temperature

Returns

Pointer to raw_temperature

Definition at line 46 of file TempSensor.cpp.

Here is the caller graph for this function:



3.7.3.5 readTempSensor_task()

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

Task for reading temperature and humidity values.

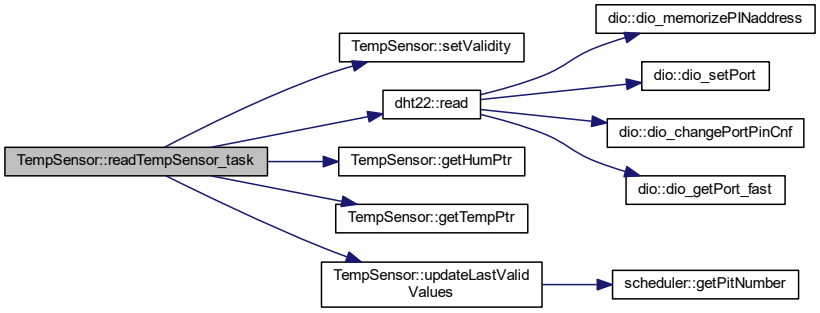
This task reads temperature and humidity data using DHT22 driver. It is called every 5 seconds.

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:

3.7.3.6 setValidity()

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

Set data val_validity.

This function sets the class member val_validity

Parameters

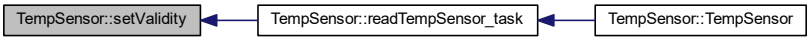
in	*validity*	Value of validity

Returns

Nothing

Definition at line 36 of file TempSensor.cpp.

Here is the caller graph for this function:

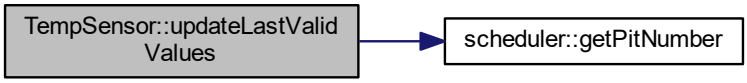


3.7.3.7 updateLastValidValues()

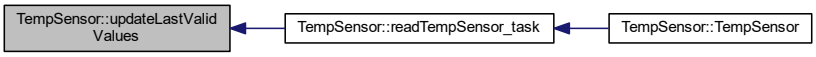
```
void TempSensor::updateLastValidValues ( )
```

Definition at line 51 of file TempSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



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

- work/asw/TempSensor/TempSensor.h
- work/asw/TempSensor/TempSensor.cpp

3.8 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.

3.8.1 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.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 timer()

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

Class constructor.

This function initializes class attributes

Returns

Nothing

Definition at line 13 of file timer.cpp.

3.8.3 Member Function Documentation

3.8.3.1 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:

3.8.3.2 startTimer1()

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

Start Timer #1.

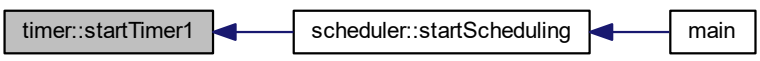
This functions starts Timer #1. Timer shall be initialized before this function is called.

Returns

Nothing

Definition at line 56 of file timer.cpp.

Here is the caller graph for this function:



3.8.3.3 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.

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

- work/bsw/timer/timer.h
- work/bsw/timer/timer.cpp

3.9 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.

3.9.1 Detailed Description

USART serial bus class.

This class defines all useful functions for USART serial bus

Definition at line 16 of file usart.h.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 usart()

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

Class usart constructor.

Initializes the class and call hardware initialization function

Parameters

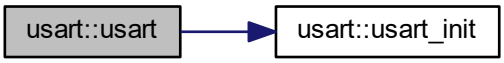
in	*a_BaudRate*	Desired Baud Rate (16 bit) - up to 57600

Returns

Nothing.

Definition at line 14 of file usart.cpp.

Here is the call graph for this function:



3.9.3 Member Function Documentation

3.9.3.1 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.

3.9.3.2 usart_init()

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

USART hardware initialization.

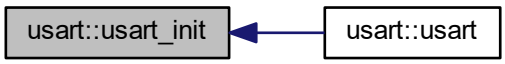
This function will initialize the USART using selected baudrate. User must pay attention to select one of the usually used Baud Rate (9600, 19200, 38400, 57600). Note that since an uint16 is used as argument, Baud rate cannot be more than 57600.

Returns

Nothing.

Definition at line 21 of file usart.cpp.

Here is the caller graph for this function:



3.9.3.3 usart_read()

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

USART read function.

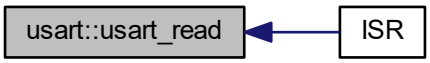
This function will read reception register of USART

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:



3.9.3.4 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

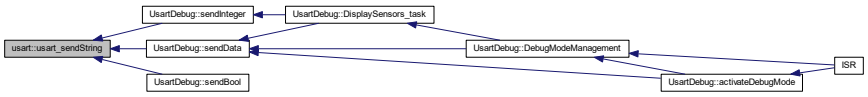
in	*str*	Pointer to the string being sent

Returns

Nothing.

Definition at line 44 of file usart.cpp.

Here is the caller graph for this function:



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

- work/bsw/usart/usart.h
- work/bsw/usart/usart.cpp

3.10 UsartDebug Class Reference

Class used for debugging on usart link.

`#include <debug.h>`

Public Member Functions

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

3.10.1 Detailed Description

Class used for debugging on usart link.

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

Definition at line 31 of file debug.h.

3.10.2 Constructor & Destructor Documentation

3.10.2.1 UsartDebug()

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

Class UsartDebug constructor.

Initializes the class UsartDebug

Returns

Nothing

Definition at line 30 of file debug.cpp.

3.10.3 Member Function Documentation

3.10.3.1 activateDebugMode()

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

Activates debug mode.

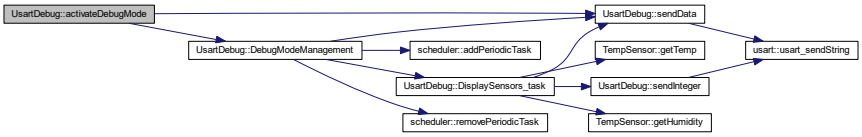
This function activates USART debug mode.

Returns

Nothing

Definition at line 114 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.10.3.2 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

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

Parameters

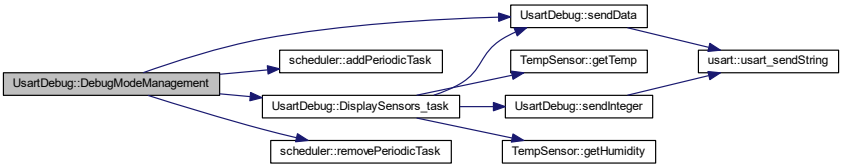
in	*rcv_char*	8 bits character received on USART

Returns

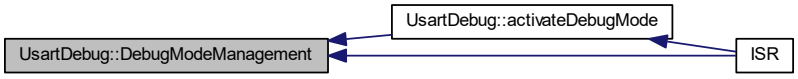
Nothing

Definition at line 122 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.10.3.3 DisplaySensors_task()

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

Displays sensors data on usart link.

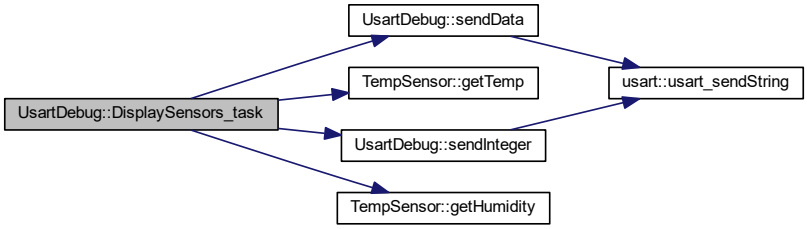
This task sends sensors data (temperature and humidity) on usart link every 5 seconds

Returns

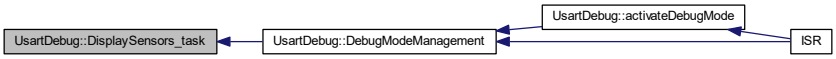
Nothing

Definition at line 68 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.10.3.4 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 109 of file debug.cpp.

Here is the caller graph for this function:

3.10.3.5 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 56 of file debug.cpp.

Here is the call graph for this function:

3.10.3.6 sendData()

```
void UsartDebug::sendData (
            char * str )
```

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

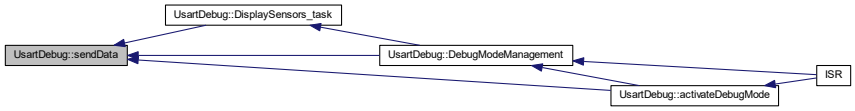
Returns

Nothing

Definition at line 36 of file debug.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



3.10.3.7 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 42 of file debug.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:

- work/asw/debug/debug.h
- work/asw/debug/debug.cpp

Chapter 4

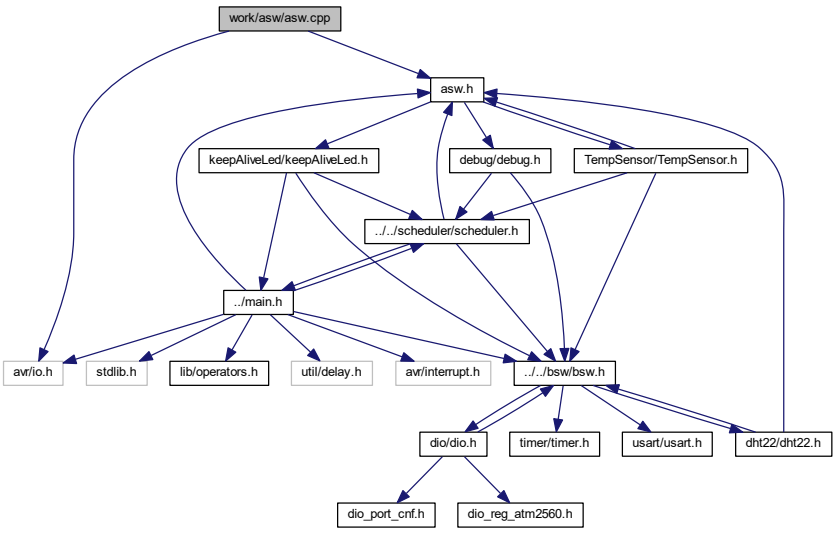
File Documentation

4.1 work/asw/asw.cpp File Reference

ASW main file.

```
#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.

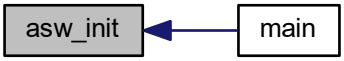
This function instantiates all applicative objects. The addresses of objects are then stored in ASW_cnf_struct structure. This function shall be called after BSW initialization function.

Returns

Nothing

Definition at line 20 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

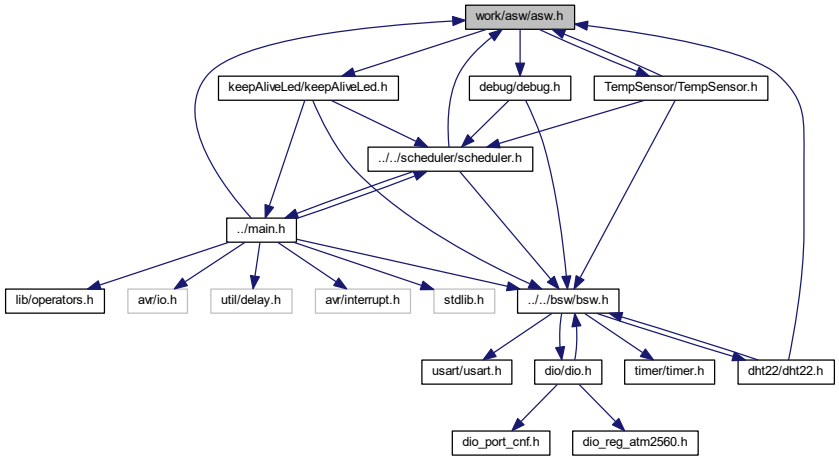
Definition at line 17 of file asw.cpp.

4.2 work/asw/asw.h File Reference

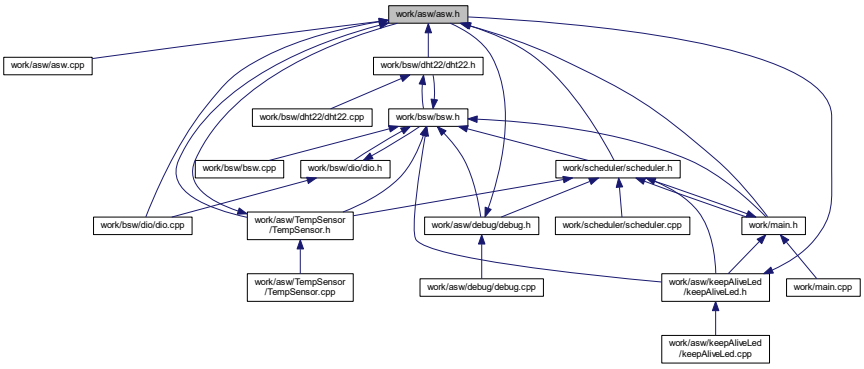
ASW main header file.

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

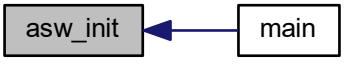
This function instantiates all applicative objects. The addresses of objects are then stored in ASW_cnf_struct structure. This function shall be called after BSW initialization function.

Returns

Nothing

Definition at line 20 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

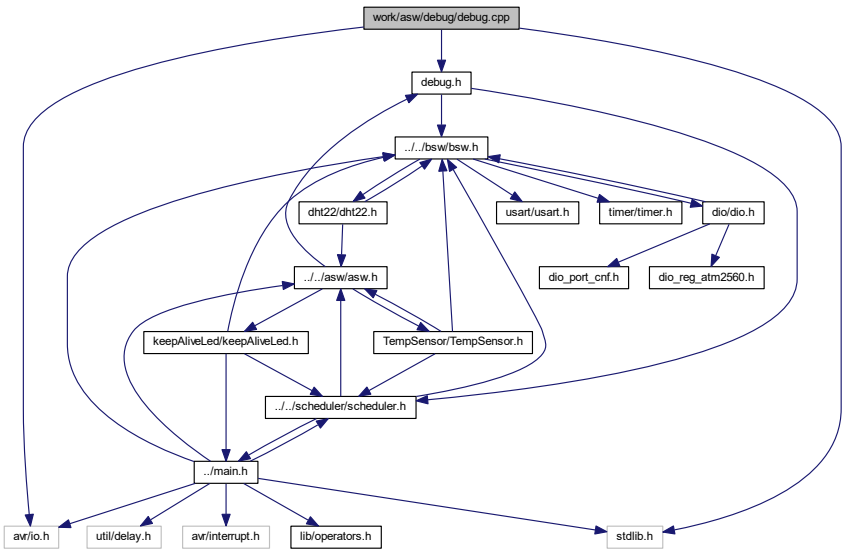
Definition at line 17 of file asw.cpp.

4.3 work/asw/debug/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.

4.3.1 Detailed Description

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

Date

15 mars 2018

Author

nicls67

4.3.2 Variable Documentation

4.3.2.1 str_debug_main_menu

```
const char str_debug_main_menu[]
```

Initial value:

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

Main menu of debug mode.

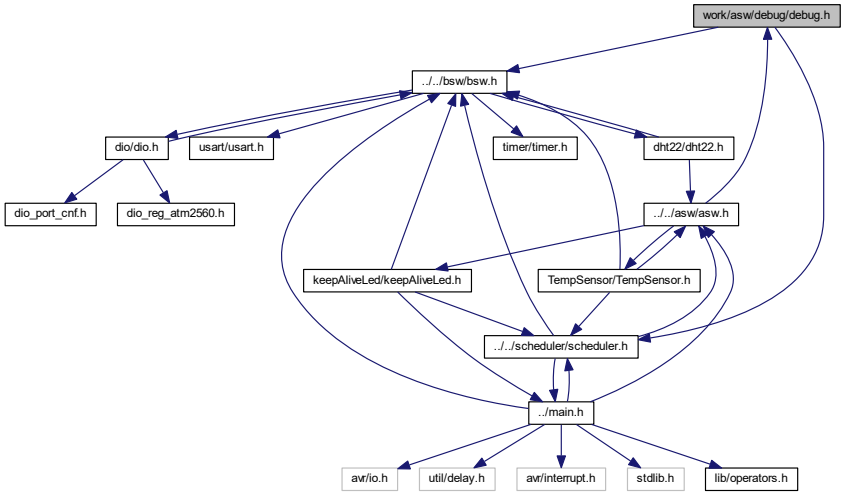
Definition at line 20 of file debug.cpp.

4.4 work/asw/debug/debug.h File Reference

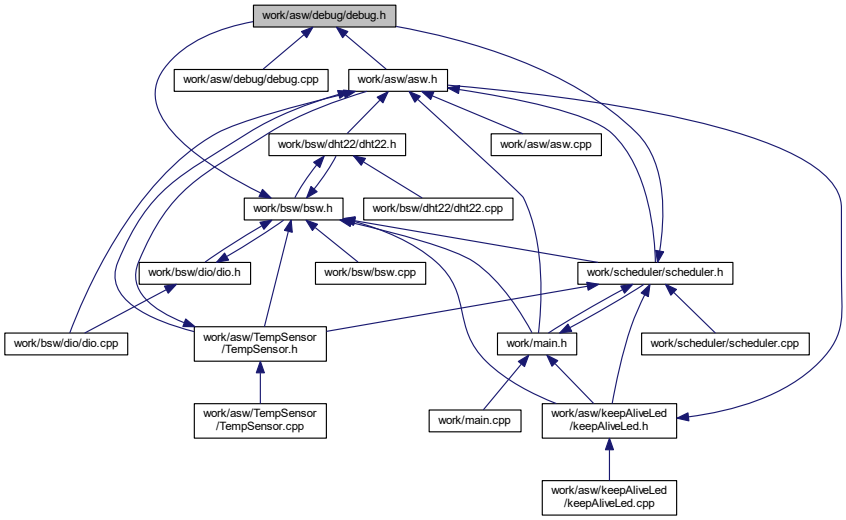
Header file for debug and logging functions.

```
#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

Enumerations

- enum debug_state_t { INIT, WAIT_INIT, DISPLAY_DATA }

  Defines the debug states.

4.4.1 Detailed Description

Header file for debug and logging functions.

Date

15 mars 2018

Author

nicls67

4.4.2 Macro Definition Documentation

4.4.2.1 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.

4.4.3 Enumeration Type Documentation

4.4.3.1 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

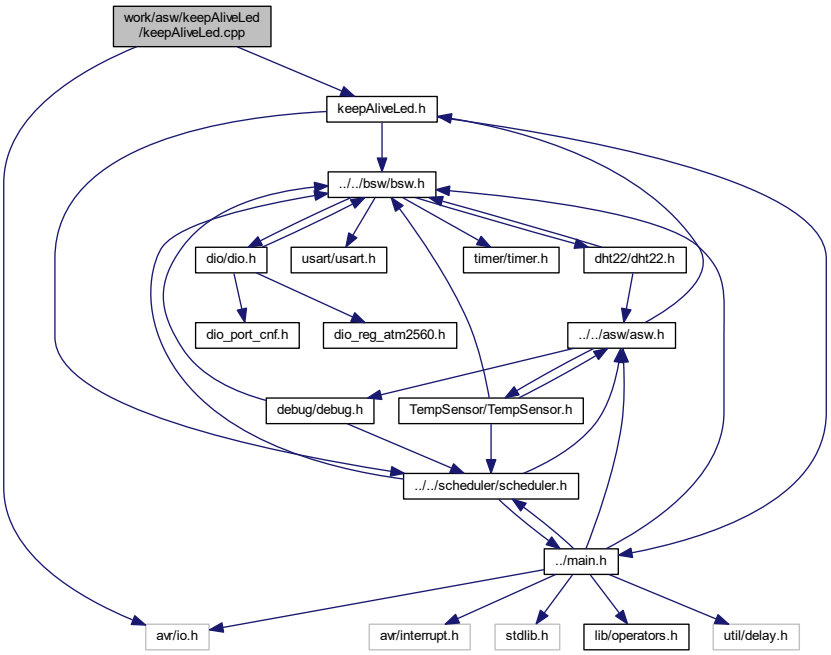
Definition at line 19 of file debug.h.

4.5 work/asw/keepAliveLed/keepAliveLed.cpp File Reference

Definition of function for class keepAliveLed.

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

Include dependency graph for keepAliveLed.cpp:



4.5.1 Detailed Description

Definition of function for class keepAliveLed.

Date

17 mars 2018

Author

nicls67

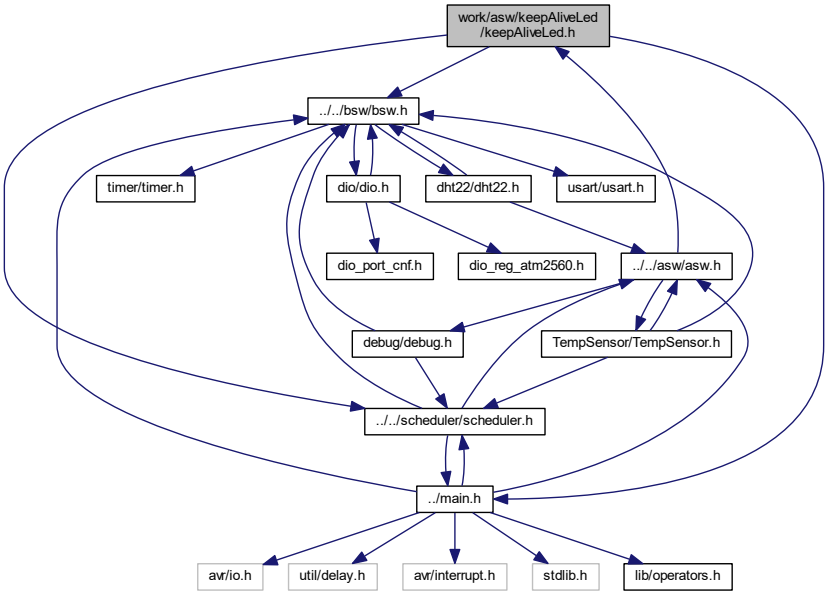
4.6 work/asw/keepAliveLed/keepAliveLed.h File Reference

Class keepAliveLed header file.

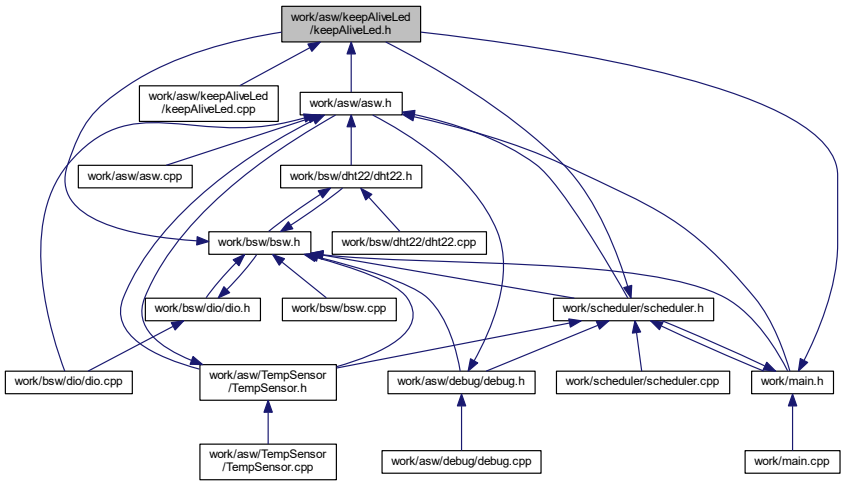
```
#include "../../bsw/bsw.h"
#include "../../scheduler/scheduler.h"
```

```
#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)

4.6.1 Detailed Description

Class keepAliveLed header file.

Date

17 mars 2018

Author

nicls67

4.6.2 Macro Definition Documentation

4.6.2.1 LED_PORT

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

LED is connected to port PB7

Definition at line 16 of file keepAliveLed.h.

4.6.2.2 PERIOD_MS_TASK_LED

```
#define PERIOD_MS_TASK_LED SW_PERIOD_MS
```

Period for led blinking

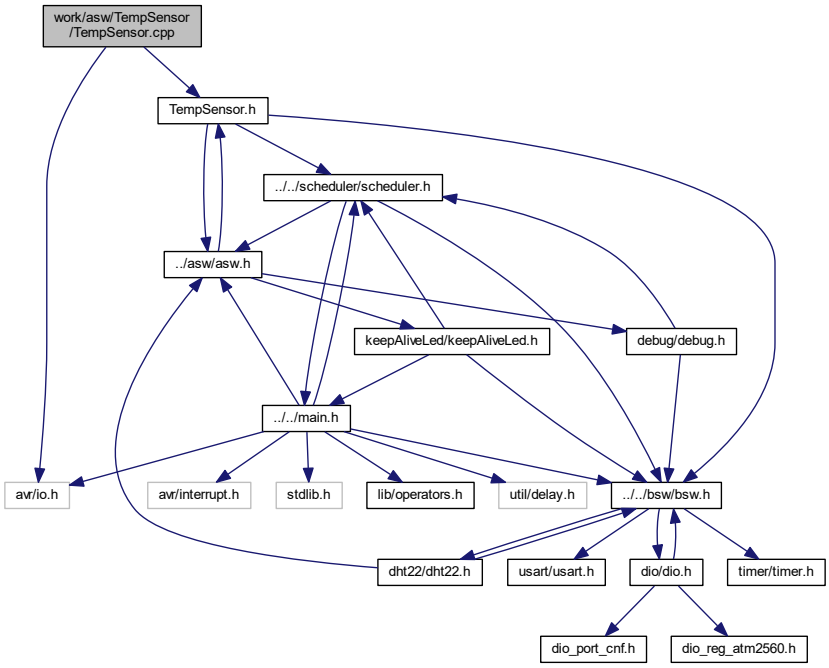
Definition at line 15 of file keepAliveLed.h.

4.7 work/asw/TempSensor/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

4.7.1 Detailed Description

Defines function of class TempSensor.

Date

23 mars 2018

Author

nicls67

4.7.2 Macro Definition Documentation

4.7.2.1 PIT_BEFORE_INVALID

```
#define PIT_BEFORE_INVALID 60
```

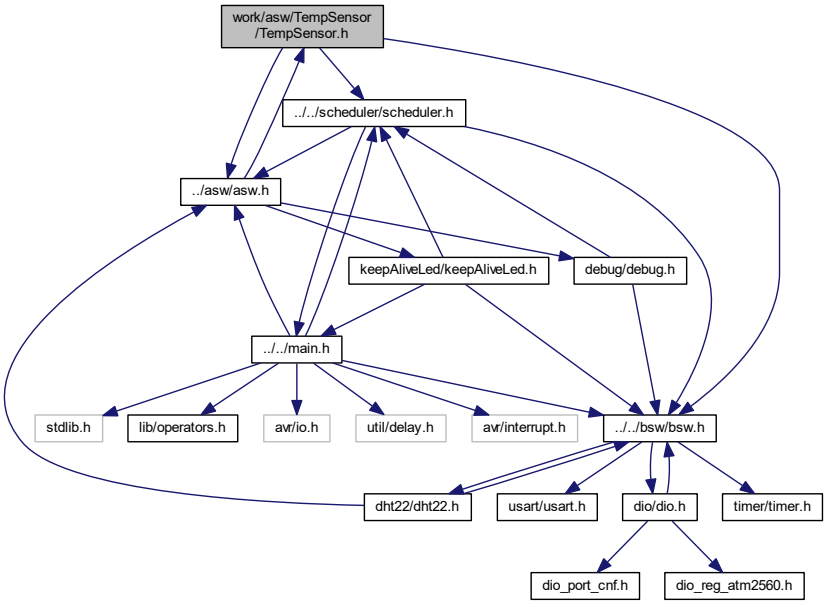
Definition at line 14 of file TempSensor.cpp.

4.8 work/asw/TempSensor/TempSensor.h File Reference

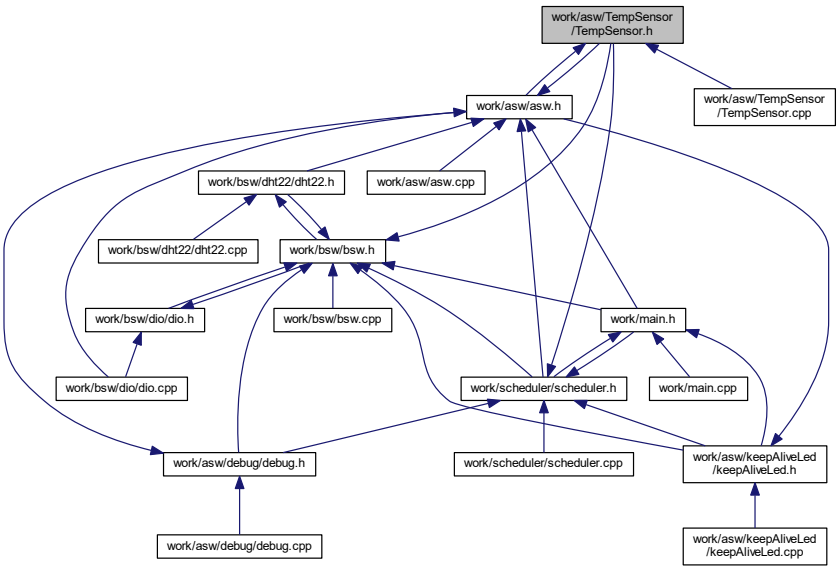
Class TempSensor header file.

```
#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

Macros

- #define PERIOD_MS_TASK_TEMP_SENSOR 5000

4.8.1 Detailed Description

Class TempSensor header file.

Date

23 mars 2018

Author

nicls67

4.8.2 Macro Definition Documentation

4.8.2.1 PERIOD_MS_TASK_TEMP_SENSOR

```
#define PERIOD_MS_TASK_TEMP_SENSOR 5000
```

Period for reading temperature data

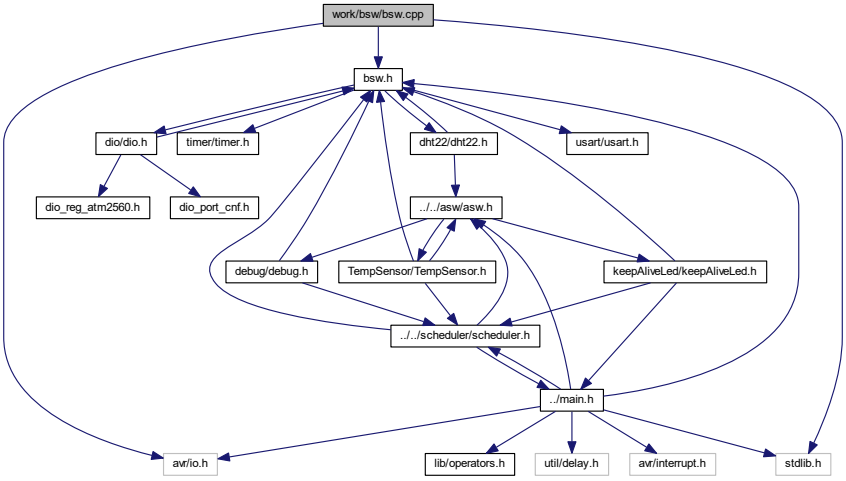
Definition at line 13 of file TempSensor.h.

4.9 work/bsw/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

4.9.1 Detailed Description

BSW main file.

Date

13 mars 2018

Author

nicls67

4.9.2 Function Documentation

4.9.2.1 bsw_init()

```
void bsw_init ( )
```

Initialization of BSW.

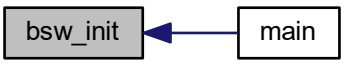
This function instantiates all driver objects, leading hardware initialization. The addresses of driver objects are then stored in BSW_cnf_struct structure.

Returns

Nothing

Definition at line 18 of file bsw.cpp.

Here is the caller graph for this function:



4.9.3 Variable Documentation

4.9.3.1 BSW_cnf_struct

```
T_BSW_cnf_struct BSW_cnf_struct
```

BSW configuration structure

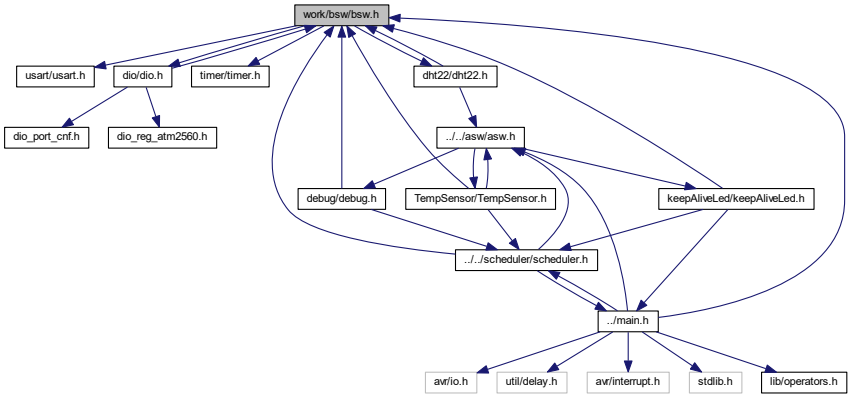
Definition at line 16 of file bsw.cpp.

4.10 work/bsw/bsw.h File Reference

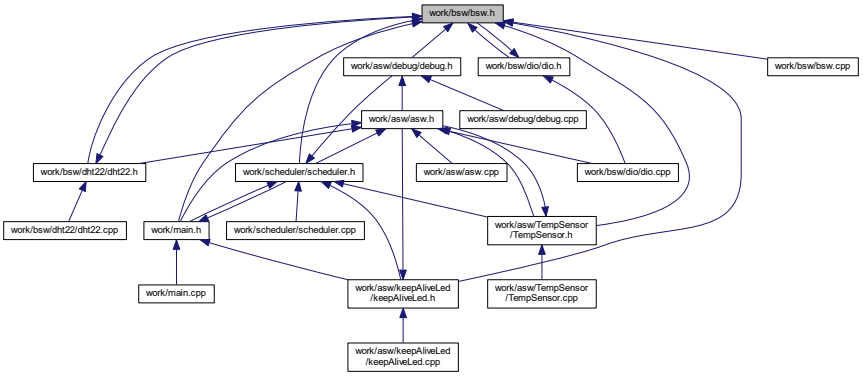
BSW main header file.

```
#include "usart/usart.h"
#include "dio/dio.h"
#include "timer/timer.h"
#include "dht22/dht22.h"
```

Include 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

Functions

- void bsw_init ()

  Initialization of BSW.

Variables

- T_BSW_cnf_struct BSW_cnf_struct

4.10.1 Detailed Description

BSW main header file.

Date

13 mars 2018

Author

nicls67

4.10.2 Macro Definition Documentation

4.10.2.1 USART_BAUDRATE

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

usart connection to PC uses a baud rate of 9600

Definition at line 23 of file bsw.h.

4.10.3 Function Documentation

4.10.3.1 bsw_init()

```
void bsw_init ( )
```

Initialization of BSW.

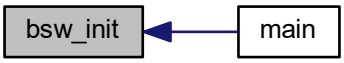
This function instantiates all driver objects, leading hardware initialization. The addresses of driver objects are then stored in BSW_cnf_struct structure.

Returns

Nothing

Definition at line 18 of file bsw.cpp.

Here is the caller graph for this function:



4.10.4 Variable Documentation

4.10.4.1 BSW_cnf_struct

```
T_BSW_cnf_struct BSW_cnf_struct
```

BSW configuration structure

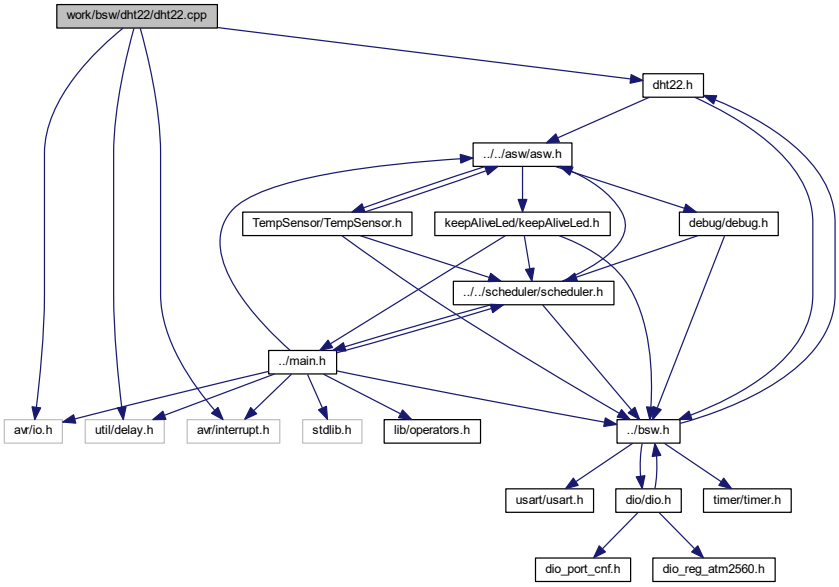
Definition at line 16 of file bsw.cpp.

4.11 work/bsw/dht22/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

4.11.1 Detailed Description

This file defines classes for DHT22 driver.

Date

23 mars 2018

Author

nicls67

4.11.2 Macro Definition Documentation

4.11.2.1 MAX_WAIT_TIME_US

```
#define MAX_WAIT_TIME_US 100
```

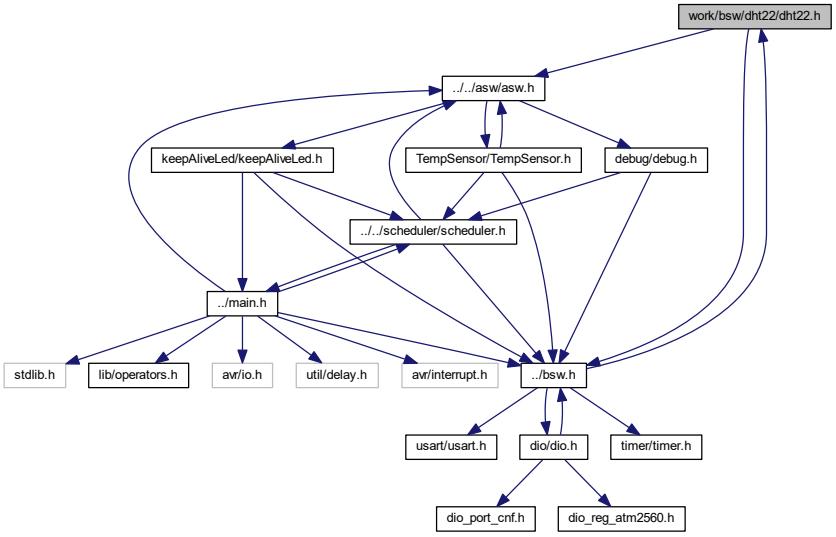
Definition at line 20 of file dht22.cpp.

4.12 work/bsw/dht22/dht22.h File Reference

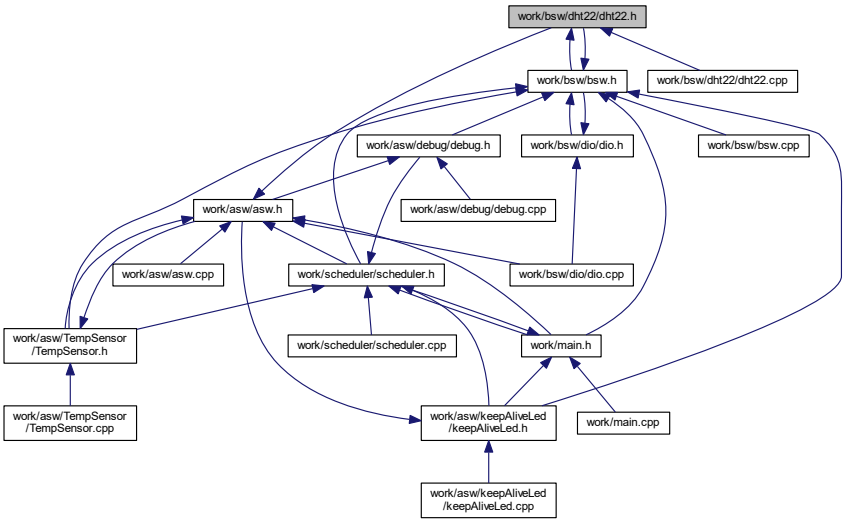
DHT22 driver header file.

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

4.12.1 Detailed Description

DHT22 driver header file.

Date

23 mars 2018

Author

nicls67

4.12.2 Macro Definition Documentation

4.12.2.1 DHT22_PORT

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

DHT22 is connected to port PB6

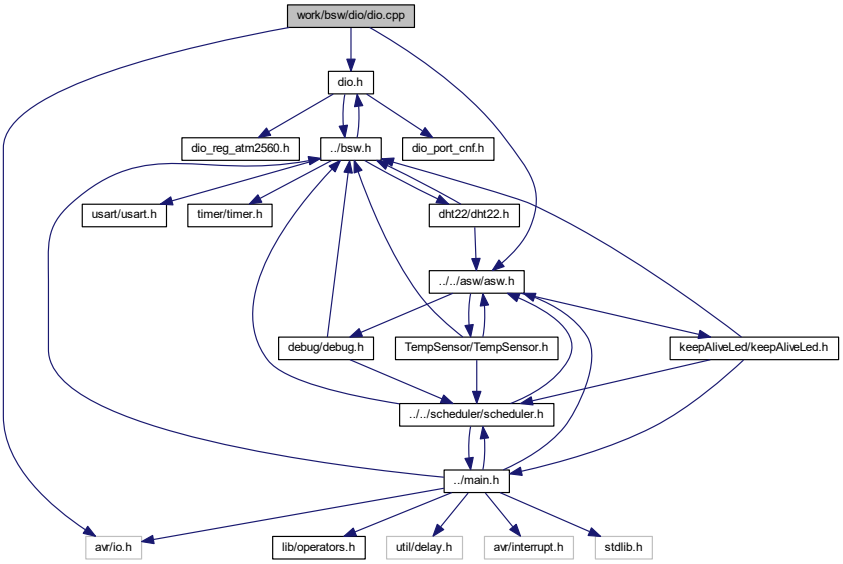
Definition at line 16 of file dht22.h.

4.13 work/bsw/dio/dio.cpp File Reference

DIO library.

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

Include dependency graph for dio.cpp:



4.13.1 Detailed Description

DIO library.

Date

13 mars 2018

Author

nicls67

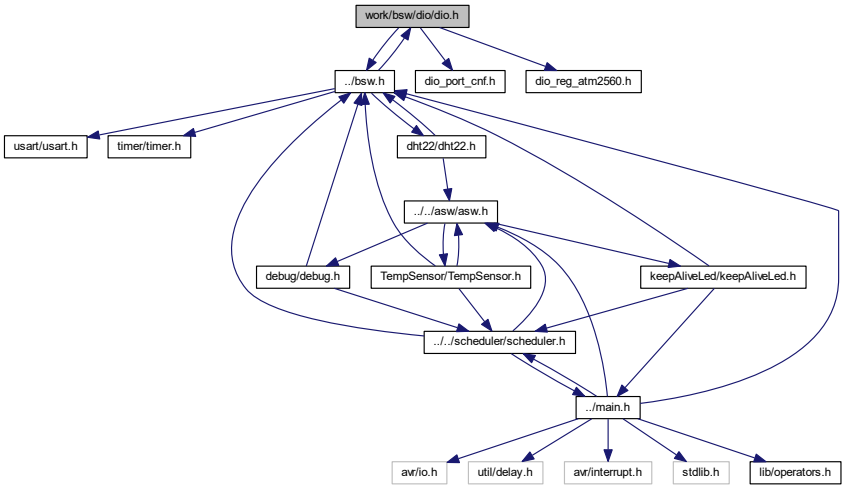
4.14 work/bsw/dio/dio.h File Reference

DIO library header file.

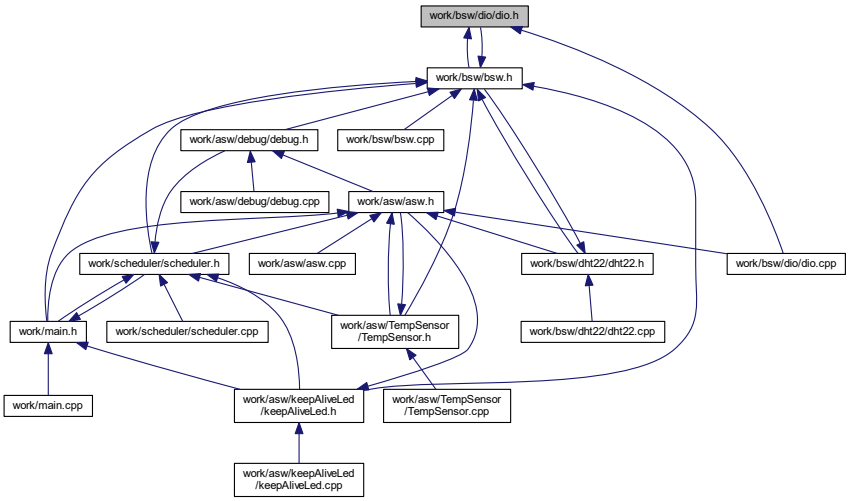
```
#include "../bsw.h"
#include "dio_port_cnf.h"
```

```
#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

4.14.1 Detailed Description

DIO library header file.

Date

13 mars 2018

Author

nicls67

4.14.2 Macro Definition Documentation

4.14.2.1 DECODE_PIN

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

Macro used to extract pin index

Definition at line 19 of file dio.h.

4.14.2.2 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.

4.14.2.3 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.

4.14.2.4 PORT_A

```
#define PORT_A 0
```

PORTA index

Definition at line 21 of file dio.h.

4.14.2.5 PORT_B

```
#define PORT_B 1
```

PORTB index

Definition at line 22 of file dio.h.

4.14.2.6 PORT_C

```
#define PORT_C 2
```

PORTC index

Definition at line 23 of file dio.h.

4.14.2.7 PORT_CNF_IN

```
#define PORT_CNF_IN 0
```

Pin is configured as input

Definition at line 15 of file dio.h.

4.14.2.8 PORT_CNF_OUT

```
#define PORT_CNF_OUT 1
```

Pin is configured as output

Definition at line 14 of file dio.h.

4.14.2.9 PORT_D

```
#define PORT_D 3
```

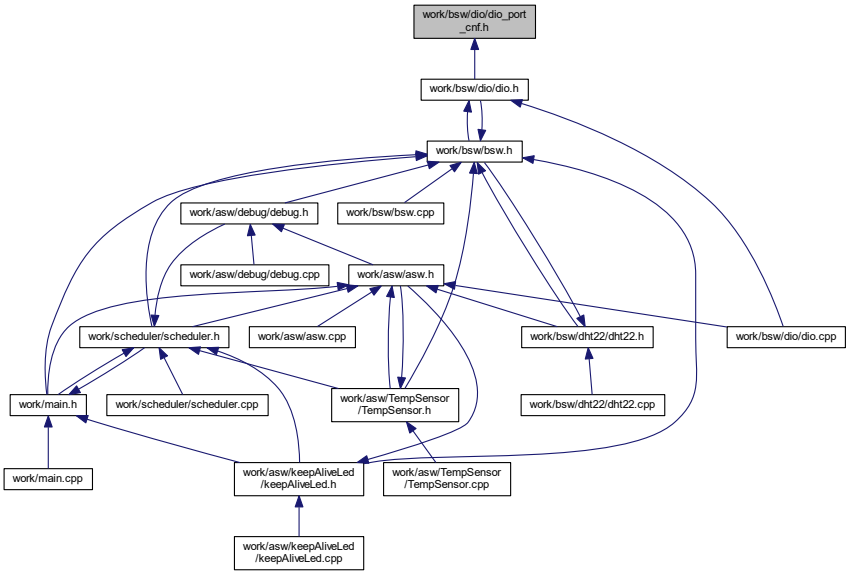
PORTD index

Definition at line 24 of file dio.h.

4.15 work/bsw/dio/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.

4.15.1 Detailed Description

Digital ports configuration file.

Date

19 mars 2019

Author

nicls67

4.15.2 Macro Definition Documentation

4.15.2.1 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.

4.15.2.2 PORTB_CNF_PORTB

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

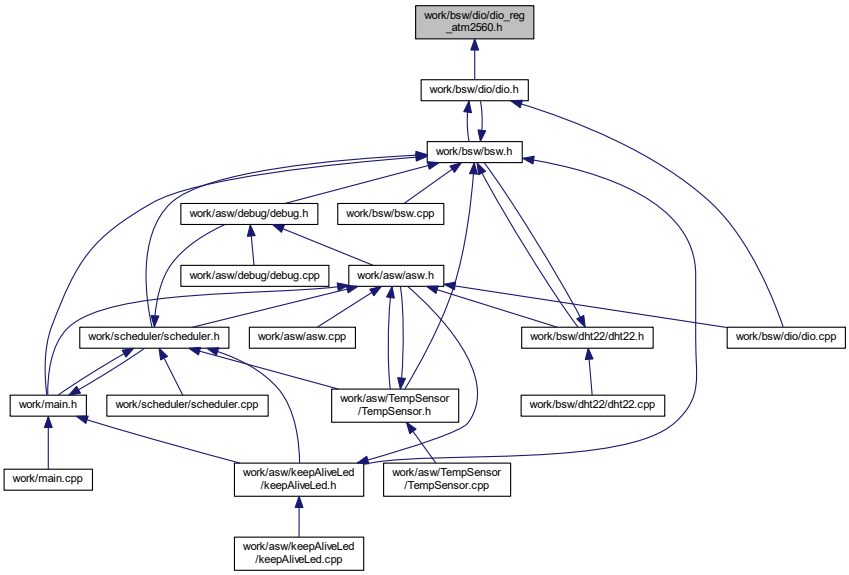
Defines the configuration of PORTB register.

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.

4.16 work/bsw/dio/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)

4.16.1 Macro Definition Documentation

4.16.1.1 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.

4.16.1.2 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.

4.16.1.3 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.

4.16.1.4 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.

4.16.1.5 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.

4.16.1.6 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.

4.16.1.7 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.

4.16.1.8 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.

4.16.1.9 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.

4.16.1.10 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.

4.16.1.11 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.

4.16.1.12 PORTD_PTR

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

Macro defining pointer to PORT D register

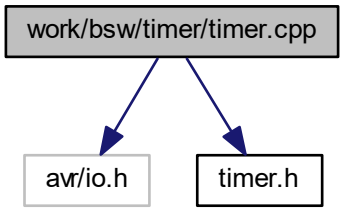
Definition at line 17 of file dio_reg_atm2560.h.

4.17 work/bsw/timer/timer.cpp File Reference

Defines function for class timer.

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

Include dependency graph for timer.cpp:



4.17.1 Detailed Description

Defines function for class timer.

Date

15 mars 2018

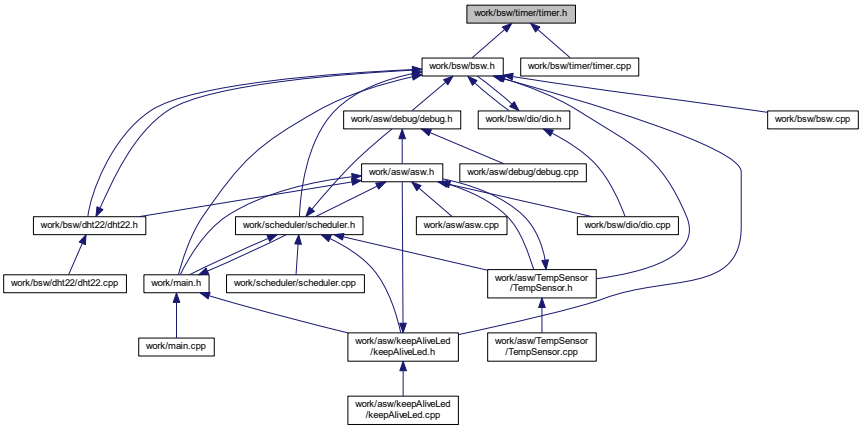
Author

nicls67

4.18 work/bsw/timer/timer.h File Reference

Timer class header file.

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



Classes

- class timer

  Class defining a timer.

4.18.1 Detailed Description

Timer class header file.

Date

15 mars 2018

Author

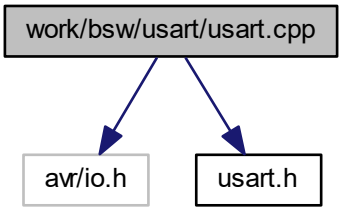
nicls67

4.19 work/bsw/usart/usart.cpp File Reference

BSW library for USART.

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

Include dependency graph for usart.cpp:



4.19.1 Detailed Description

BSW library for USART.

Date

13 mars 2018

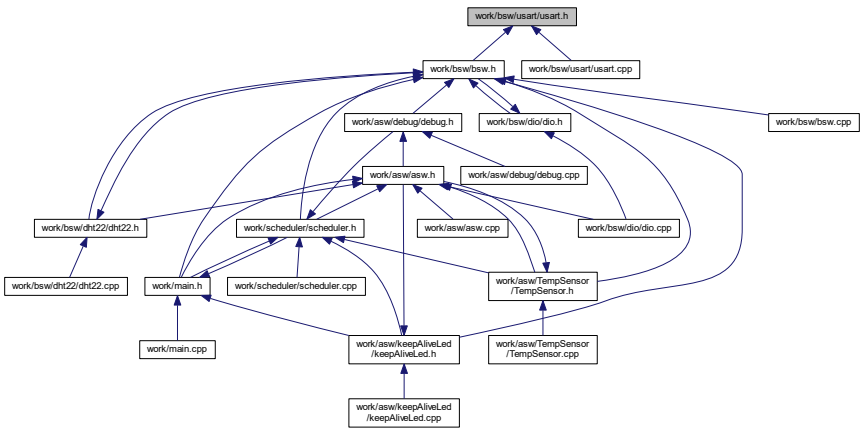
Author

nicls67

4.20 work/bsw/usart/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.

4.20.1 Detailed Description

Header file for USART library.

Date

13 mars 2018

Author

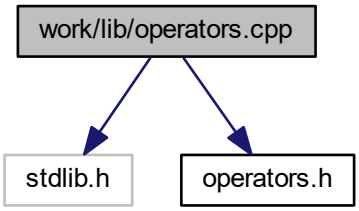
nicls67

4.21 work/lib/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.

4.21.1 Detailed Description

c++ operators definitions

Date

14 mars 2018

Author

nicls67

4.21.2 Function Documentation

4.21.2.1 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.

4.21.2.2 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

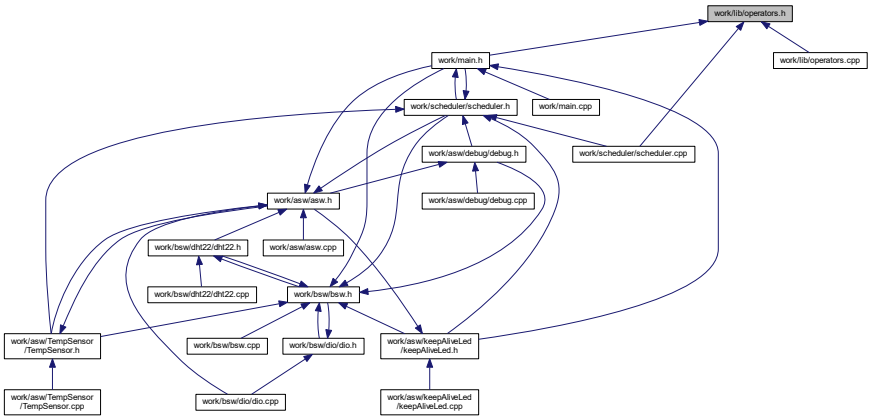
Pointer to the start of allocated memory zone

Definition at line 13 of file operators.cpp.

4.22 work/lib/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.

4.22.1 Detailed Description

c++ operators definitions header file

Date

14 mars 2018

Author

nicls67

4.22.2 Function Documentation

4.22.2.1 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.

4.22.2.2 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

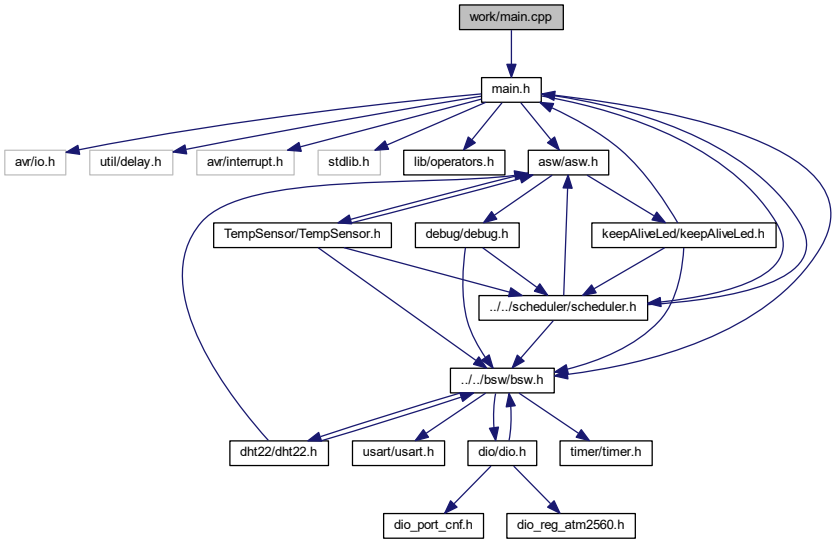
Definition at line 13 of file operators.cpp.

4.23 work/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.

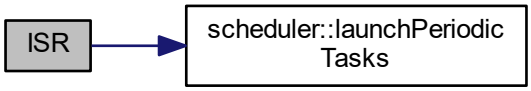
This function handles the interrupt raised by Timer #1. It wakes up the software every 500 ms to perform applications.

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.

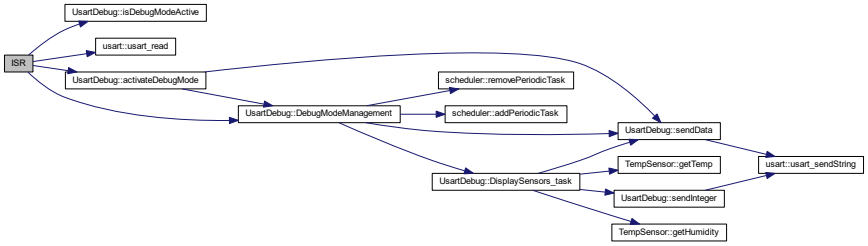
This function handles the interrupt raised when a frame has been received by USART. If debug mode mode is active, it calls debug mode management function. If inactive, it calls debug mode activation function if the received character is 'a'

Returns

Nothing

Definition at line 31 of file main.cpp.

Here is the call graph for this function:



4.23.2.3 main()

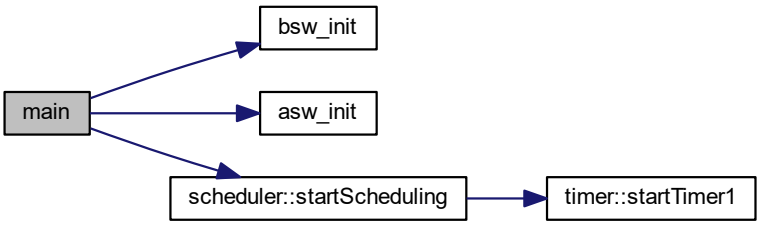
```
int main (
            void  )
```

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:

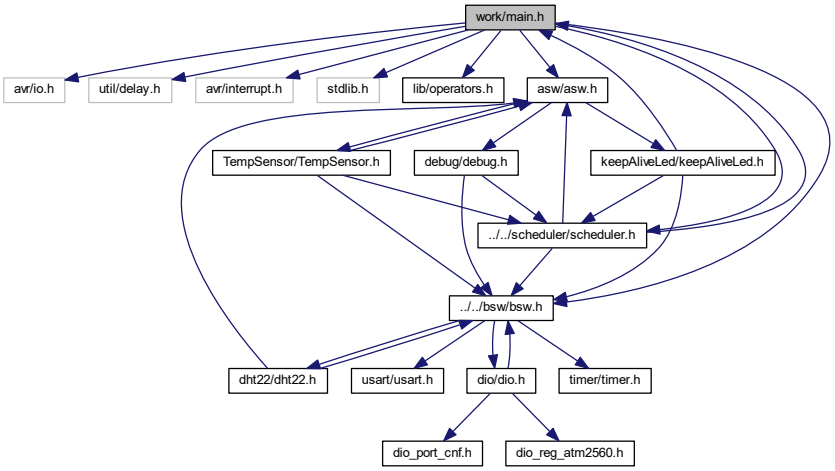


4.24 work/main.h File Reference

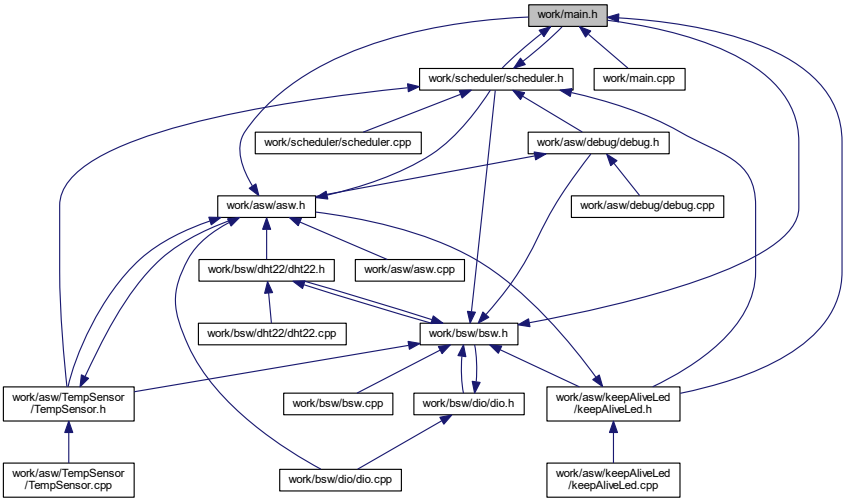
Background task header file.

```
#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

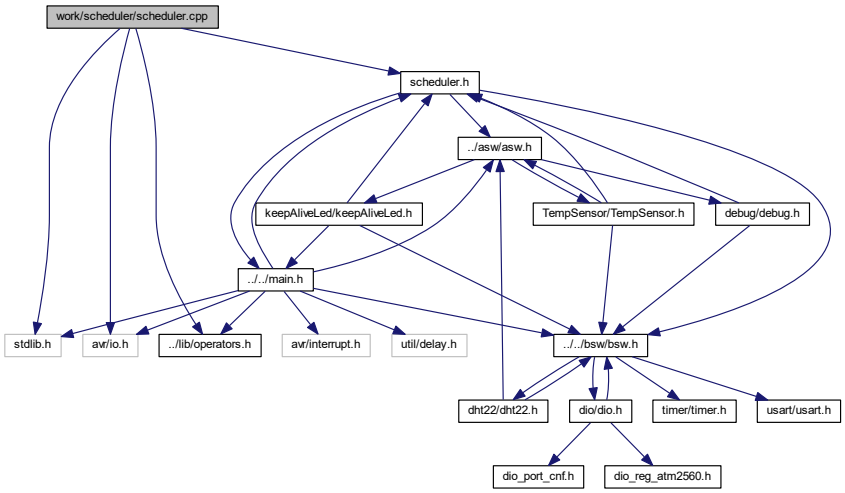
nicls67

4.25 work/scheduler/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

4.25.1 Detailed Description

Defines scheduler class.

Date

16 mars 2018

Author

nicls67

4.25.2 Variable Documentation

4.25.2.1 p_scheduler

```
scheduler* p_scheduler
```

Pointer to scheduler object

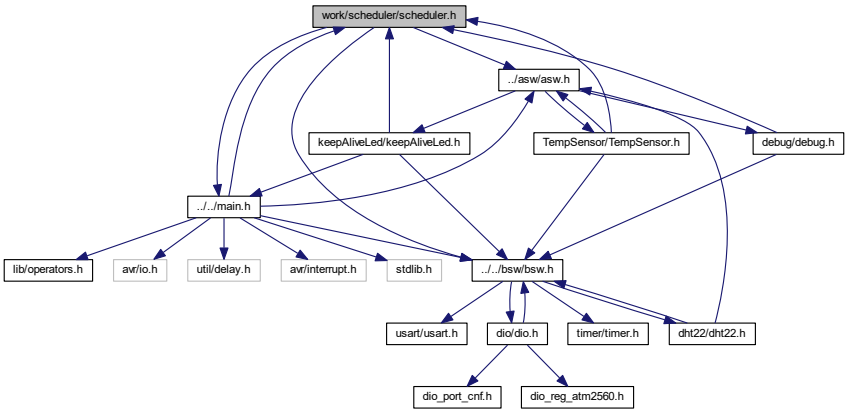
Definition at line 17 of file scheduler.cpp.

4.26 work/scheduler/scheduler.h File Reference

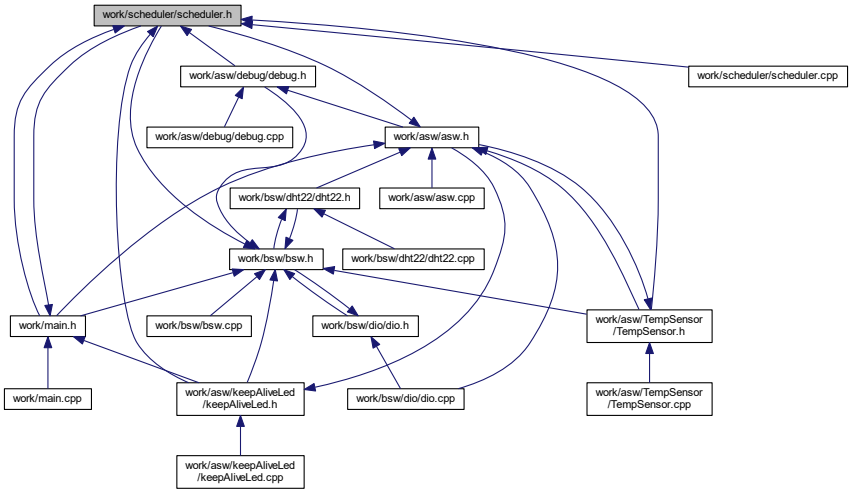
Scheduler class header file.

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

Include dependency graph for scheduler.h:



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



Classes

- class scheduler

  Scheduler class.

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

4.26.1 Detailed Description

Scheduler class header file.

Date

16 mars 2018

Author

nicls67

4.26.2 Macro Definition Documentation

4.26.2.1 PRESCALER_PERIODIC_TIMER

```
#define PRESCALER_PERIODIC_TIMER 256
```

Value of prescaler to use for periodic timer

Definition at line 19 of file scheduler.h.

4.26.2.2 SW_PERIOD_MS

```
#define SW_PERIOD_MS 500
```

Software period, used to define periodic timer interrupt

Definition at line 18 of file scheduler.h.

4.26.2.3 TIMER_CTC_VALUE

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

Compare value for periodic timer

Definition at line 20 of file scheduler.h.

4.26.3 Typedef Documentation

4.26.3.1 TaskPtr_t

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

Type defining a pointer to function.

Definition at line 25 of file scheduler.h.

4.26.4 Variable Documentation

4.26.4.1 p_scheduler

```
scheduler* p_scheduler
```

Pointer to scheduler object

Definition at line 17 of file scheduler.cpp.

Index