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. It also creates a new object of Timer class in case it is still not created. Normally the CpuLoad class is used by the scheduler object, which should create the Timer object. Thus the initialization of Timer object in CpuLoad class should not be needed. We still do the check here to avoid any issue with null pointer.

Returns

Nothing

Definition at line 20 of file CpuLoad.cpp.

3.1.3 Member Function Documentation

3.1.3.1 ComputeCPULoad()

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

Computes current CPU load.

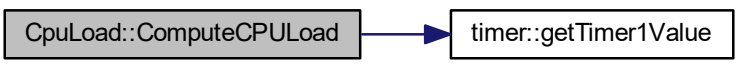
This function computes the current CPU load using value of the timer used by the scheduler at the end of the periodic cycle. This value is divided by the PIT period to obtain CPU load;

Returns

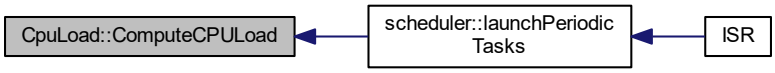
Nothing

Definition at line 40 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 58 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 47 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.

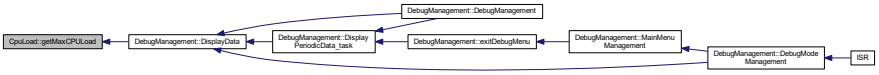
This function returns the maximum CPU load value

Returns

Maximum CPU load value

Definition at line 69 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 76 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 75 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 81 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 77 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 78 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 80 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 79 of file CpuLoad.h.

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

- CpuLoad.h
- CpuLoad.cpp

3.2 debug_mgt_state_struct_t Struct Reference

Structure containing all debug states.

`#include <DebugManagement.h>`

Public Attributes

- debug_mgt_main_menu_state_t main_state
- debug_mgt_wdg_state_t wdg_state

3.2.1 Detailed Description

Structure containing all debug states.

Definition at line 40 of file DebugManagement.h.

3.2.2 Member Data Documentation

3.2.2.1 main_state

`debug_mgt_main_menu_state_t debug_mgt_state_struct_t::main_state`

Current main menu state

Definition at line 42 of file DebugManagement.h.

3.2.2.2 wdg_state

`debug_mgt_wdg_state_t debug_mgt_state_struct_t::wdg_state`

Current state of watchdog management

Definition at line 43 of file DebugManagement.h.

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

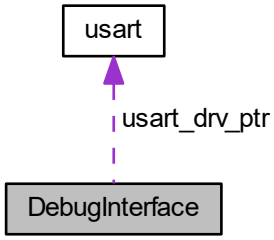
- DebugManagement.h

3.3 DebugInterface Class Reference

Class used for debugging on usart link.

`#include <DebugInterface.h>`

Collaboration diagram for DebugInterface:



Public Member Functions

- DebugInterface ()
  Class DebugInterface constructor.
- void sendInteger (uint16_t data, uint8_t base)
  Send a integer data on USART link.
- void sendBool (bool data, bool isText)
  Send a boolean data on USART link.
- void sendString (String *str)
  Send a string on USART link.
- void sendString (uint8_t *str)
  Send a chain of characters on USART link.
- void sendChar (uint8_t chr)
  Send a single character on USART link.
- uint8_t read ()
  USART read function.
- void nextLine ()
  Go to next line function.
- void ClearScreen ()
  Screen clearing function.

Private Attributes

- usart * usart_drv_ptr

3.3.1 Detailed Description

Class used for debugging on usart link.

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

Definition at line 21 of file DebugInterface.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 DebugInterface()

```
DebugInterface::DebugInterface ( )
```

Class DebugInterface constructor.

Initializes the class DebugInterface. It creates a new instance of USART driver of needed.

Returns

Nothing

Definition at line 22 of file DebugInterface.cpp.

3.3.3 Member Function Documentation

3.3.3.1 ClearScreen()

```
void DebugInterface::ClearScreen ( )
```

Screen clearing function.

This function clears the entire display by sending the character on the USART line.

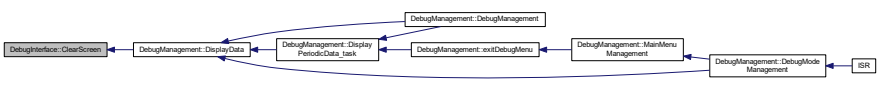
Returns

Nothing

Definition at line 77 of file DebugInterface.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



3.3.3.2 nextLine()

```
void DebugInterface::nextLine ( )
```

Go to next line function.

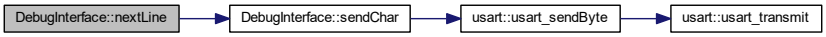
This function goes to the next line on the console display. It sends the two characters and on the USART line.

Returns

Nothing

Definition at line 71 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.3.3.3 read()

```
uint8_t DebugInterface::read ( )  [inline]
```

USART read function.

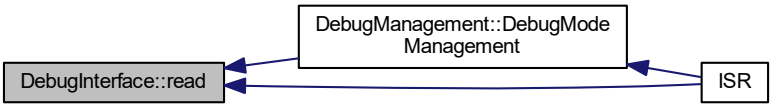
This function will read the last received byte on USART link

Returns

Received byte

Definition at line 82 of file DebugInterface.h.

Here is the caller graph for this function:



3.3.3.4 sendBool()

```
void DebugInterface::sendBool (
            bool data,
            bool isText )
```

Send a boolean data on USART link.

This function 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. The parameter isText defines if the data is converted into a string (true/false) or an integer (1/0).

Parameters

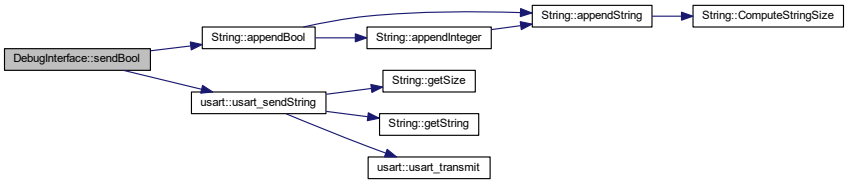
in	*data*	boolean data to be sent
in	*isText*	String conversion configuration

Returns

Nothing

Definition at line 62 of file DebugInterface.cpp.

Here is the call graph for this function:



3.3.3.5 sendChar()

```
void DebugInterface::sendChar (
            uint8_t chr )
```

Send a single character on USART link.

This function sends the requested character on USART link by calling driver's transmission function.

Parameters

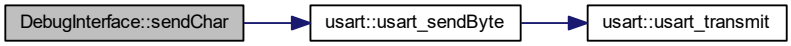
in	*chr*	Character to send.

Returns

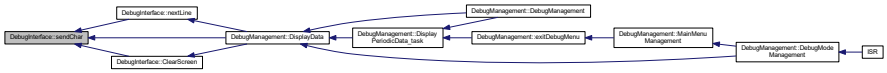
Nothing

Definition at line 44 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.3.3.6 sendInteger()

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

Send a integer data on USART link.

This function 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

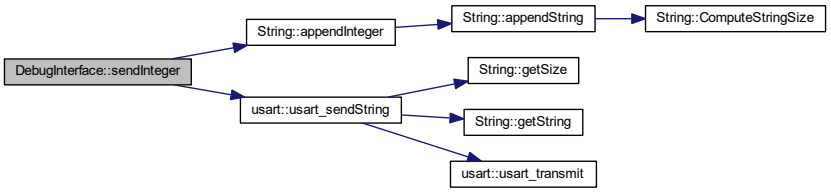
in	*data*	integer data to be sent
in	*base*	numerical base used to convert integer into string (between 2 and 36)

Returns

Nothing

Definition at line 49 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.3.3.7 sendString() [1/2]

```
void DebugInterface::sendString (
            String * str )
```

Send a string on USART link.

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

Parameters

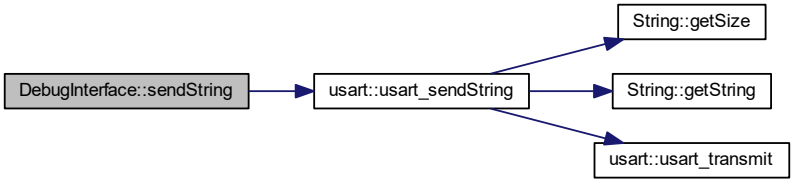
in	*str*	Pointer to the string being sent

Returns

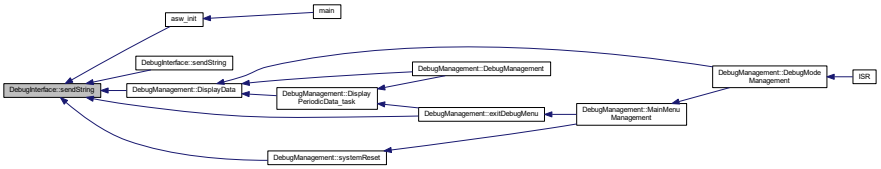
Nothing

Definition at line 31 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.3.3.8 sendString() [2/2]

```
void DebugInterface::sendString (
            uint8_t * str )
```

Send a chain of characters on USART link.

This function sends the requested chain of characters on USART link by calling driver's transmission function. The chain is first converted into a string object.

Parameters

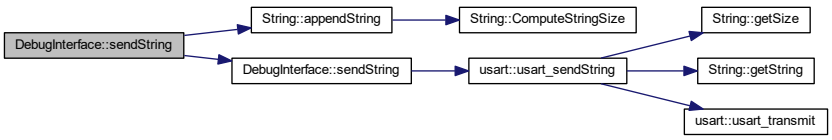
in	*str*	Pointer to the chain to send.

Returns

Nothing

Definition at line 37 of file DebugInterface.cpp.

Here is the call graph for this function:



3.3.4 Member Data Documentation

3.3.4.1 usart_drv_ptr

```
usart* DebugInterface::usart_drv_ptr  [private]
```

Pointer to USART driver object

Definition at line 107 of file DebugInterface.h.

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

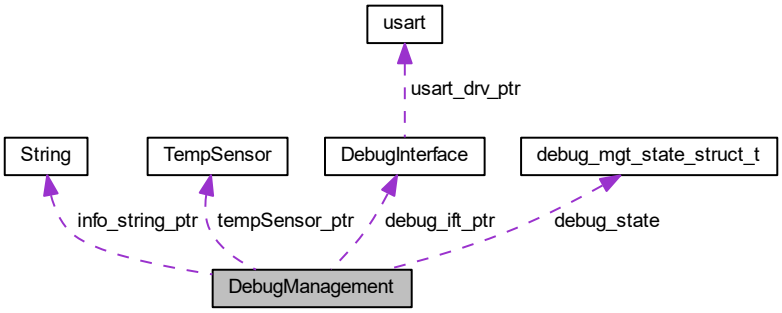
- DebugInterface.h
- DebugInterface.cpp

3.4 DebugManagement Class Reference

Debug management class.

```
#include <DebugManagement.h>
```

Collaboration diagram for DebugManagement:



Public Member Functions

- DebugManagement ()
  Class constructor.
- void DisplayData ()
  Displays data on usart link.
- bool DebugModeManagement ()
  Management of debug mode.
- DebugInterface * getIftPtr ()
  Interface pointer get function.
- uint8_t * getMenuStringPtr ()
  Menu string get function.
- String * getInfoStringPtr ()
  Info string get function.
- void setInfoStringPtr (String *addr)
  Info message setting function.

Static Public Member Functions

- static void DisplayPeriodicData_task ()

  Displays periodic data on usart link.

Private Member Functions

- void exitDebugMenu ()

  Debug menu exit function.
- void systemReset ()

  System reset function.
- void WatchdogMenuManagement (uint8_t rcv_char)

  Watchdog menu management function.
- bool MainMenuManagement (uint8_t rcv_char)

  Main menu management.

Private Attributes

- DebugInterface * debug_ift_ptr
- TempSensor * tempSensor_ptr
- uint8_t * menu_string_ptr
- String * info_string_ptr
- debug_mgt_state_struct_t debug_state
- bool isInfoStringDisplayed

3.4.1 Detailed Description

Debug management class.

This class manages the debug menu available on USART interface. It allows to display SW informations like sensors data, CPU load...

Definition at line 51 of file DebugManagement.h.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 DebugManagement()

```
DebugManagement::DebugManagement ( )
```

Class constructor.

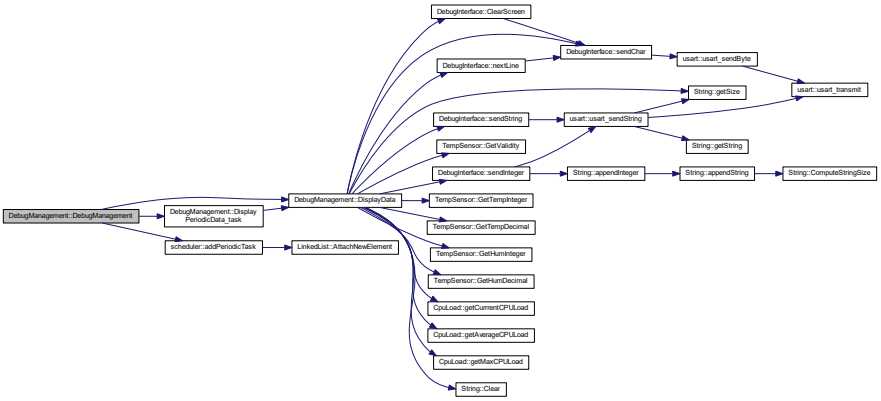
This function initializes the class. If needed, it creates a new instance of debug interface object.

Returns

Nothing

Definition at line 85 of file DebugManagement.cpp.

Here is the call graph for this function:



3.4.3 Member Function Documentation

3.4.3.1 DebugModeManagement()

```
bool DebugManagement::DebugModeManagement ( )
```

Management of debug mode.

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

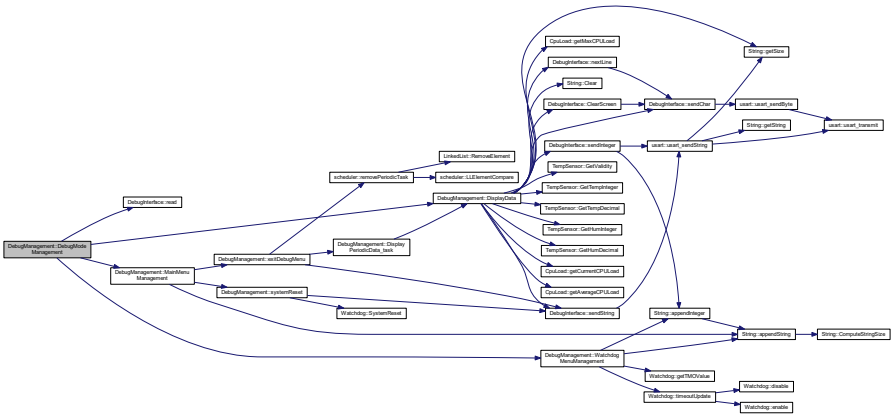
- MAIN_MENU state : handles user choice in main menu and selects next state
- WDG_MENU state : handles user choice in watchdog menu and selects next state
  It is called each time a data is received on USART and debug mode is active.

Returns

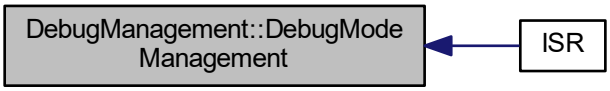
True if the debug mode shall be closed, false otherwise

Definition at line 205 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.2 DisplayData()

```
void DebugManagement::DisplayData ( )
```

Displays data on usart link.

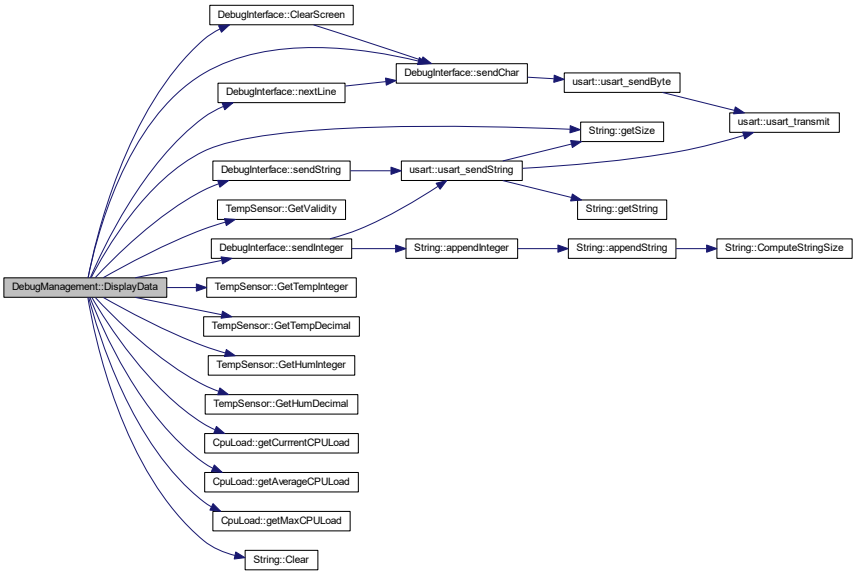
This task displays the menu and periodic data (temperature, humidity and CPU load) on usart screen.

Returns

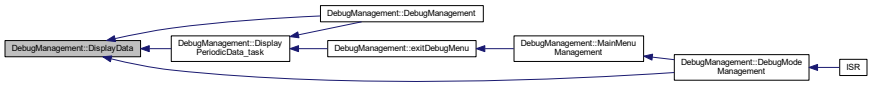
Nothing

Definition at line 114 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.3 DisplayPeriodicData_task()

```
void DebugManagement::DisplayPeriodicData_task ( )  [static]
```

Displays periodic data on usart link.

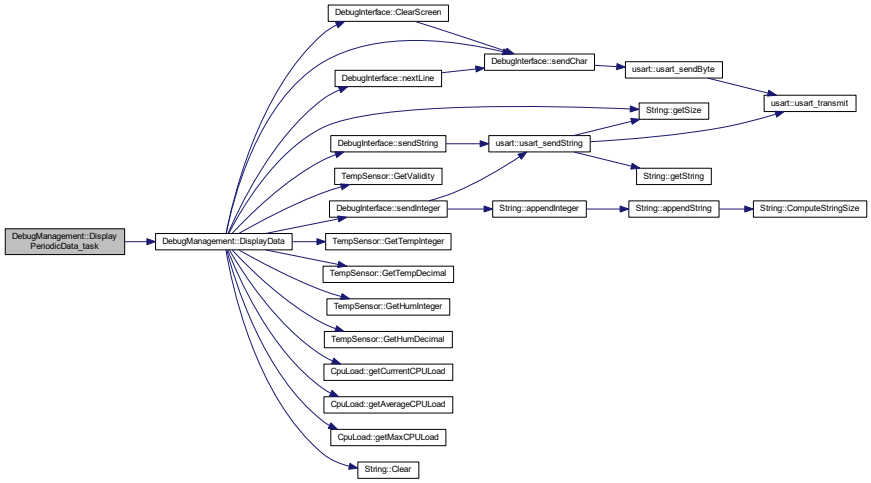
This task displays the menu and periodic data (temperature, humidity and CPU load) on usart screen. It only calls the function DisplayData.

Returns

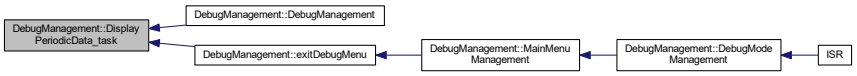
Nothing

Definition at line 197 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.4 exitDebugMenu()

```
void DebugManagement::exitDebugMenu ( )  [private]
```

Debug menu exit function.

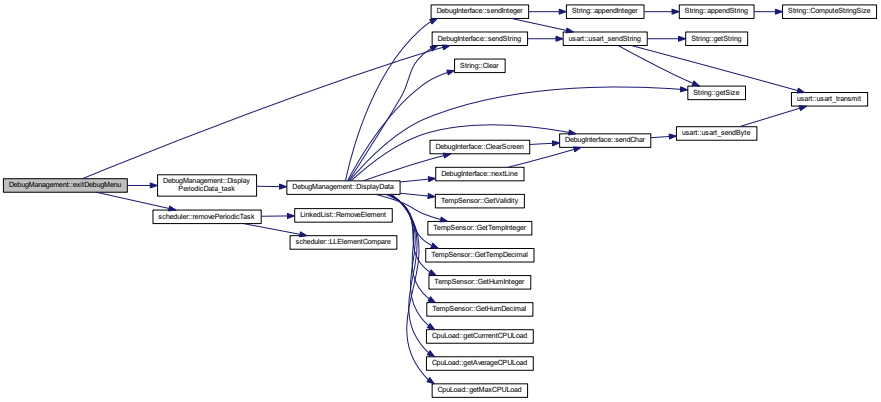
This function prepares the exit of debug menu. It writes the message "Bye !" on the screen and removes the periodic task from the scheduler.

Returns

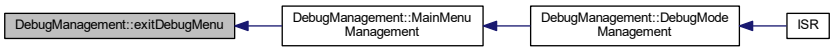
Nothing.

Definition at line 232 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.5 getIftPtr()

```
DebugInterface* DebugManagement::getIftPtr ( )  [inline]
```

Interface pointer get function.

This function returns the pointer to the debug interface object

Returns

Pointer to debug interface

Definition at line 95 of file DebugManagement.h.

3.4.3.6 getInfoStringPtr()

```
String* DebugManagement::getInfoStringPtr ( )  [inline]
```

Info string get function.

This function returns the pointer to the info string to display

Returns

Info string pointer

Definition at line 115 of file DebugManagement.h.

3.4.3.7 getMenuStringPtr()

```
uint8_t* DebugManagement::getMenuStringPtr ( )  [inline]
```

Menu string get function.

This function returns the pointer to the menu string to display

Returns

Menu string pointer

Definition at line 105 of file DebugManagement.h.

3.4.3.8 MainMenuManagement()

```
bool DebugManagement::MainMenuManagement (
            uint8_t rcv_char )  [private]
```

Main menu management.

This function manages the main debug menu. It handles the character received on USART bus and execute the requested action. It also manages the display of the main menu.

Parameters

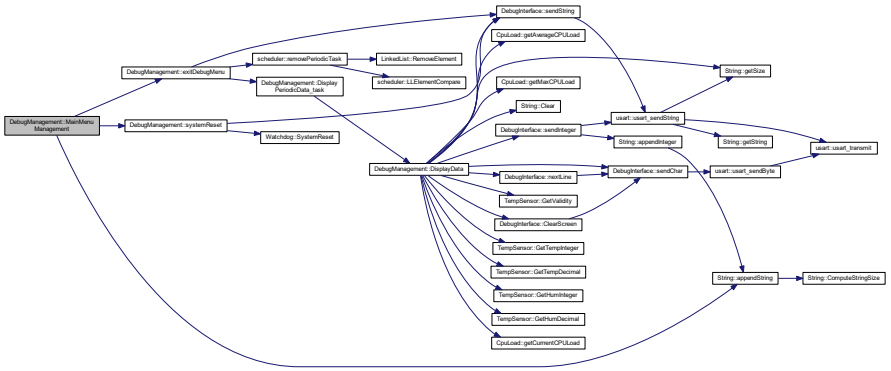
in	*rcv_char*	Character received on USART bus.

Returns

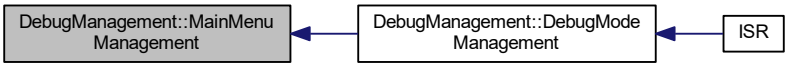
True if the debug mode shall be exited, false otherwise.

Definition at line 325 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.9 setInfoStringPtr()

```
void DebugManagement::setInfoStringPtr (
            String * addr )  [inline]
```

Info message setting function.

This functions sets the info message pointer to the given string address

Parameters

in	*addr*	String address

Returns

Nothing

Definition at line 126 of file DebugManagement.h.

3.4.3.10 systemReset()

```
void DebugManagement::systemReset ( )  [private]
```

System reset function.

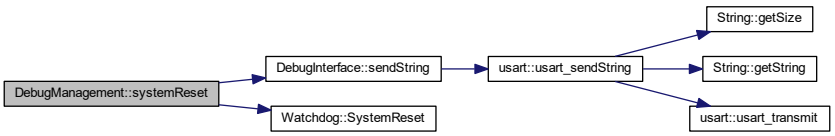
This function provokes a reset of the system. It displays a message on the screen and calls the reset function from watchdog class.

Returns

Nothing.

Definition at line 238 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.4.3.11 WatchdogMenuManagement()

```
void DebugManagement::WatchdogMenuManagement (
            uint8_t rcv_char )  [private]
```

Watchdog menu management function.

This function manages the watchdog menu. It handles the character received on USART bus and execute the requested action. It also manages the display of the watchdog menu.

Parameters

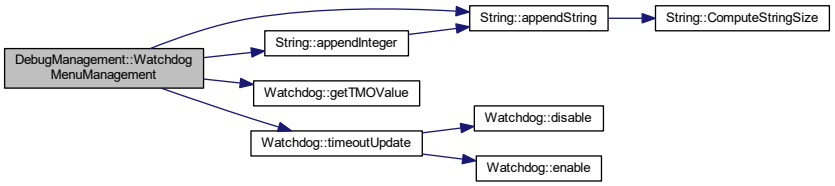
in	*rcv_char*	Character received on USART bus.

Returns

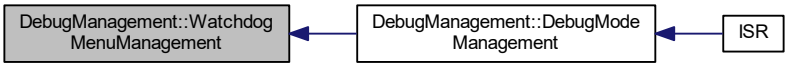
Nothing.

Definition at line 244 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.4 Member Data Documentation

3.4.4.1 debug_ift_ptr

```
DebugInterface* DebugManagement::debug_ift_ptr  [private]
```

Pointer to the debug interface object, which is used to send data on usart link

Definition at line 133 of file DebugManagement.h.

3.4.4.2 debug_state

```
debug_mgt_state_struct_t DebugManagement::debug_state  [private]
```

Structure containing debug states for each menu

Definition at line 137 of file DebugManagement.h.

3.4.4.3 info_string_ptr

```
String* DebugManagement::info_string_ptr  [private]
```

Pointer to the info message to display

Definition at line 136 of file DebugManagement.h.

3.4.4.4 isInfoStringDisplayed

```
bool DebugManagement::isInfoStringDisplayed  [private]
```

Value defining if the info string has been already displayed one complete cycle of not

Definition at line 138 of file DebugManagement.h.

3.4.4.5 menu_string_ptr

```
uint8_t* DebugManagement::menu_string_ptr  [private]
```

Pointer to the current menu string to display

Definition at line 135 of file DebugManagement.h.

3.4.4.6 tempSensor_ptr

```
TempSensor* DebugManagement::tempSensor_ptr  [private]
```

Pointer to the temperature sensor object

Definition at line 134 of file DebugManagement.h.

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

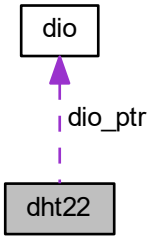
- DebugManagement.h
- DebugManagement.cpp

3.5 dht22 Class Reference

DHT 22 driver class.

`#include <dht22.h>`

Collaboration diagram for dht22:



Public Member Functions

- dht22 (uint8_t port)
  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.

Private Attributes

- uint8_t dht22_port
- dio * dio_ptr

3.5.1 Detailed Description

DHT 22 driver class.

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

Definition at line 19 of file dht22.h.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 dht22()

```
dht22::dht22 (
            uint8_t port )
```

dht22 class constructor.

Initializes the class dht22.

Parameters

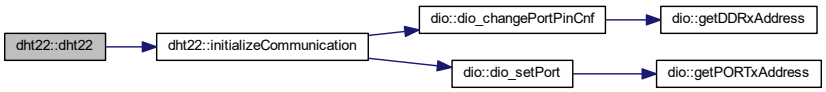
in	*port*	Encoded configuration of the port used for 1-wire communication.

Returns

Nothing

Definition at line 22 of file dht22.cpp.

Here is the call graph for this function:



3.5.3 Member Function Documentation

3.5.3.1 initializeCommunication()

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

Initializes the communication.

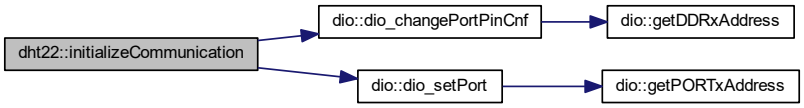
This function initializes the communication with DHT22 using 1-wire protocol

Returns

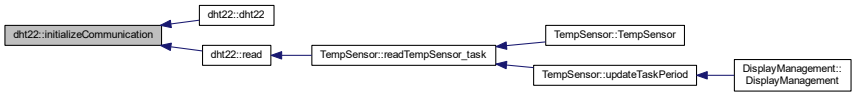
Nothing

Definition at line 201 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.5.3.2 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

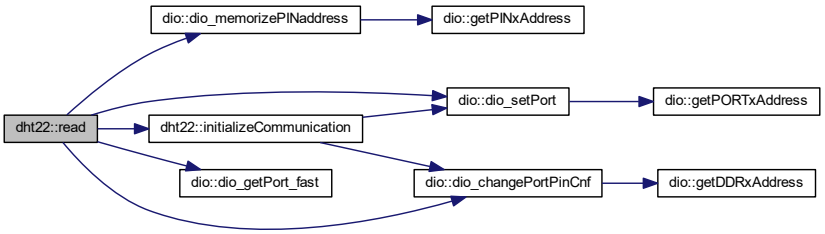
out	*raw_humidity*	Raw humidity value received from sensor
out	*raw_temperature*	Raw temperature value received from sensor

Returns

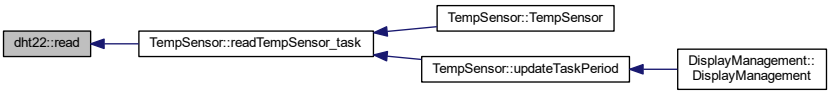
Validity of the read value

Definition at line 30 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.5.4 Member Data Documentation

3.5.4.1 dht22_port

```
uint8_t dht22::dht22_port  [private]
```

Variable containing the port used for 1-wire communication

Definition at line 45 of file dht22.h.

3.5.4.2 dio_ptr

```
dio* dht22::dio_ptr  [private]
```

Pointer to the DIO object

Definition at line 46 of file dht22.h.

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

- dht22.h
- dht22.cpp

3.6 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

3.6.1 Detailed Description

DIO class.

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

Definition at line 27 of file dio.h.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 dio()

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

dio class constructor

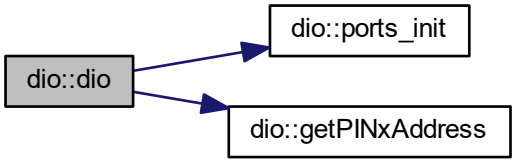
Initializes class dio and calls DIO hardware initialization function

Returns

Nothing

Definition at line 112 of file dio.cpp.

Here is the call graph for this function:



3.6.3 Member Function Documentation

3.6.3.1 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

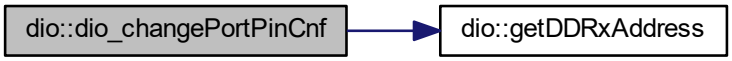
in	*portcode*	Encoded pin and register index
in	*cnf*	Requested configuration for the selected pin PORT_CNF_OUT (1) : pin configured as output PORT_CNF_IN (0) : pin configured as input

Returns

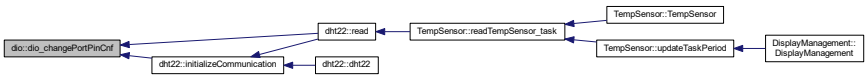
Nothing

Definition at line 149 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.6.3.2 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

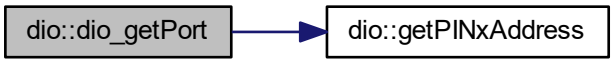
in	*portcode*	Encoded pin and register index

Returns

Logical state of selected pin

Definition at line 139 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.6.3.3 dio_getPort_fast()

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

Gets the logical state of the memorized pin.

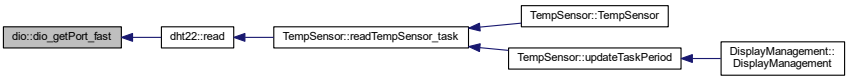
This function gets the logical value of the memorized pin. The corresponding port and pin index are stored in class members PINx_addr_mem and PINx_idx_mem. This mechanism is used to speed up reading time as this function no longer needs to extract register address and pin index from portcode.

Returns

Logical state of selected pin

Definition at line 171 of file dio.cpp.

Here is the caller graph for this function:



3.6.3.4 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

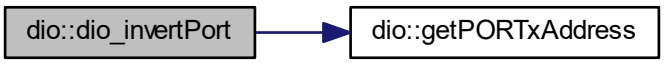
in	*portcode*	Encoded pin and register index

Returns

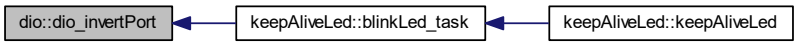
Nothing

Definition at line 131 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.6.3.5 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

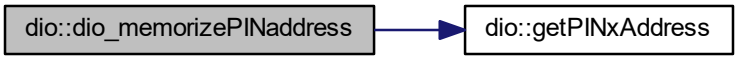
in	*portcode*	Encoded pin and register index

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:

3.6.3.6 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

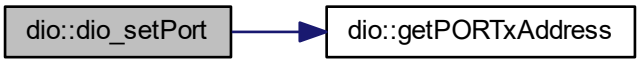
in	*portcode*	Encoded pin and register index
in	*state*	Requested state to set pin

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:

3.6.3.7 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:

3.6.3.8 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

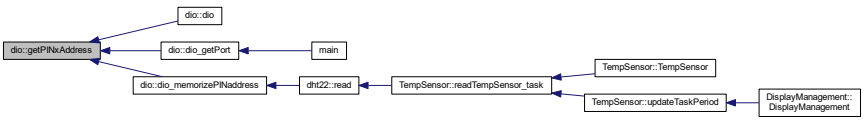
in	*portcode*	Encoded port code

Returns

Pointer to the PINx register

Definition at line 54 of file dio.cpp.

Here is the caller graph for this function:



3.6.3.9 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:

3.6.3.10 ports_init()

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

Digital ports hardware initialization function.

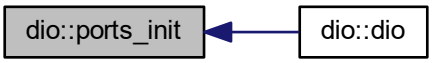
This function initializes digital ports as input or output and sets their initial values

Returns

Nothing

Definition at line 16 of file dio.cpp.

Here is the caller graph for this function:



3.6.4 Member Data Documentation

3.6.4.1 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 142 of file dio.h.

3.6.4.2 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 143 of file dio.h.

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

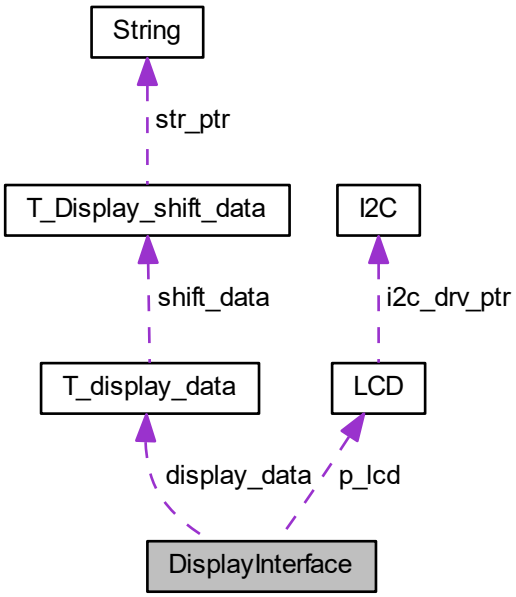
- dio.h
- dio.cpp

3.7 DisplayInterface Class Reference

Display interface services class.

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

Collaboration diagram for DisplayInterface:



Public Member Functions

- DisplayInterface (const T_LCD_conf_struct *LCD_init_cnf)

  Class constructor.
- bool DisplayFullLine (uint8_t *str, uint8_t size, uint8_t line, T_DisplayInterface_LineDisplayMode mode=NORMAL, T_DisplayInterface_LineAlignment alignment=LEFT)

  Line display function.
- bool ClearLine (uint8_t line)

  Line clearing function.
- void ClearFullScreen ()

  Screen cleaning function.
- bool IsLineEmpty (uint8_t line)

  Empty line get function.
- T_display_data * getDisplayDataPtr ()

  Display data get function.
- void setLineAlignmentAndRefresh (uint8_t line, T_DisplayInterface_LineAlignment alignment)

  Text alignment function.

Static Public Member Functions

- static void shiftLine_task ()

  Line shifting periodic task.

Private Member Functions

- uint8_t FindFirstCharAddr (uint8_t line)

  Finds start address of a line.
- void RefreshLine (uint8_t line)

  Line refresh function.
- void ClearStringInDataStruct (uint8_t line)

  String data clearing structure.
- void setLineAlignment (uint8_t line)

  Text alignment setting function.
- void updateLineAndRefresh (uint8_t *str, uint8_t size, uint8_t line)

  Line data string update function.

Private Attributes

- LCD * p_lcd
- uint32_t dummy
- T_display_data display_data [LCD_SIZE_NB_LINES]
- bool isShiftInProgress

3.7.1 Detailed Description

Display interface services class.

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

Definition at line 76 of file DisplayInterface.h.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 DisplayInterface()

```
DisplayInterface::DisplayInterface (
            const T_LCD_conf_struct * LCD_init_cnf )
```

Class constructor.

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

Parameters

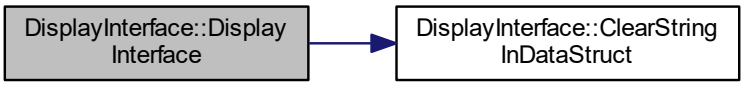
in	*LCD_init_cnf*	Initial configuration of the screen

Returns

Nothing

Definition at line 27 of file DisplayInterface.cpp.

Here is the call graph for this function:



3.7.3 Member Function Documentation

3.7.3.1 ClearFullScreen()

```
void DisplayInterface::ClearFullScreen ( )
```

Screen cleaning function.

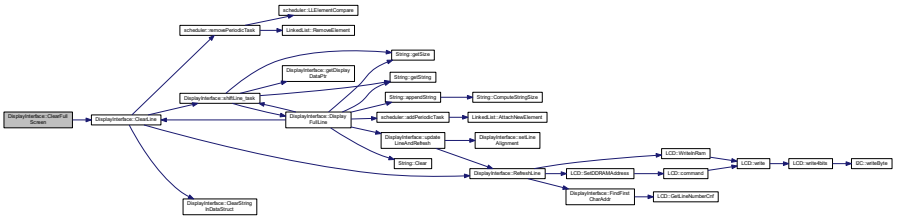
This functions clears the entire display. It uses the ClearLine function on every line of the screen.

Returns

Nothing

Definition at line 265 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.7.3.2 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. If it was the last line with a display shift in progress, it removes the periodic task from the scheduler.

Parameters

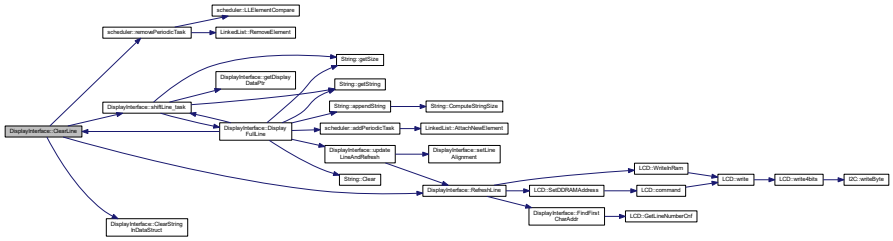
in	*line*	Line to clear

Returns

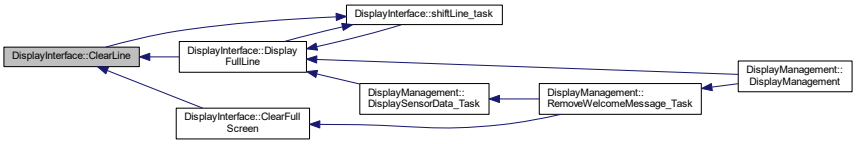
True if the line has been cleared, false otherwise

Definition at line 221 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.3.3 ClearStringInDataStruct()

```
void DisplayInterface::ClearStringInDataStruct (
            uint8_t line )  [private]
```

String data clearing structure.

This function clears the string contained in the display data structure. It sets all characters to space character.

Parameters

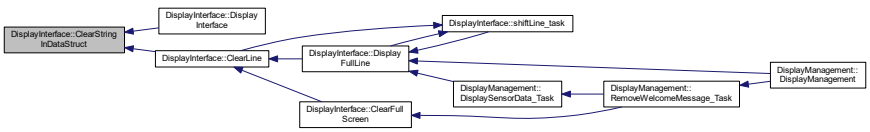
in	*line*	Line to clear

Returns

Nothing

Definition at line 174 of file DisplayInterface.cpp.

Here is the caller graph for this function:



3.7.3.4 DisplayFullLine()

```
bool DisplayInterface::DisplayFullLine (
            uint8_t * str,
            uint8_t size,
            uint8_t line,
            T_DisplayInterface_LineDisplayMode mode = NORMAL,
            T_DisplayInterface_LineAlignment alignment = LEFT )
```

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

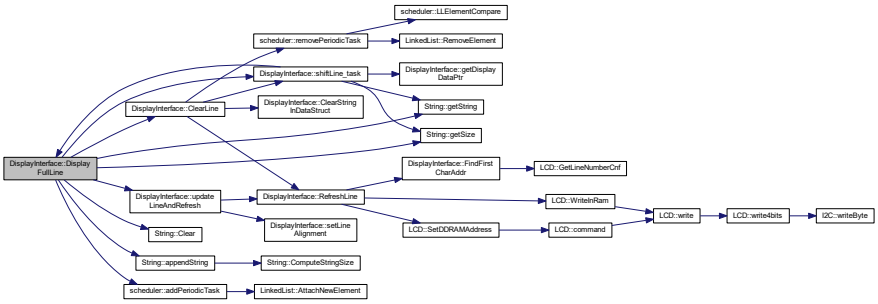
in	*str*	Pointer to the string to display
in	*size*	Size of the string to display
in	*line*	Index of the line where the string shall be displayed
in	*mode*	Display mode
in	*alignment*	Requested alignment for the line

Returns

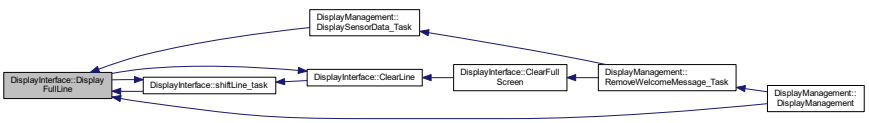
True if the line has been correctly displayed, false otherwise

Definition at line 59 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.3.5 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

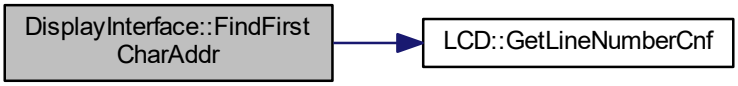
in	*line*	Line which address shall be found

Returns

Address in DDRAM of the first character of the line

Definition at line 182 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.7.3.6 getDisplayDataPtr()

```
T_display_data* DisplayInterface::getDisplayDataPtr ( )  [inline]
```

Display data get function.

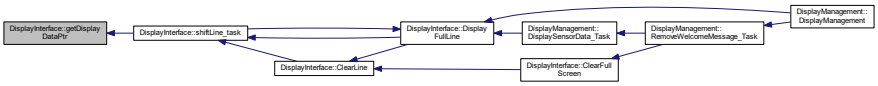
This function returns a pointer to the display data structure.

Returns

Pointer to display data structure.

Definition at line 142 of file DisplayInterface.h.

Here is the caller graph for this function:



3.7.3.7 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 273 of file DisplayInterface.cpp.

3.7.3.8 RefreshLine()

```
void DisplayInterface::RefreshLine (
            uint8_t line )  [private]
```

Line refresh function.

This function refreshes the display on the requested line. It computes the screen RAM address and writes the string to display into the screen RAM. It shall be called everytime the string in display data structure is updated.

Parameters

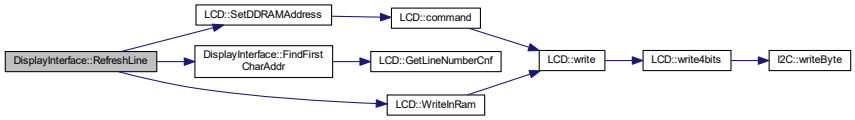
in	*line*	Line to refresh

Returns

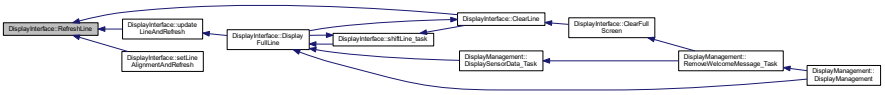
Nothing

Definition at line 161 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.3.9 setLineAlignment()

```
void DisplayInterface::setLineAlignment (
            uint8_t line )  [private]
```

Text alignment setting function.

This function updates the text alignment on the requested line. The string in the data structure is updated with the new alignment. The alignment parameter in the data structure shall be updated before calling this function.

Parameters

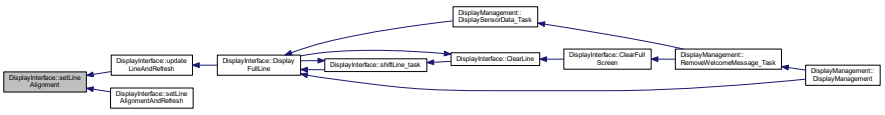
in	*line*	Line to update

Returns

Nothing

Definition at line 330 of file DisplayInterface.cpp.

Here is the caller graph for this function:



3.7.3.10 setLineAlignmentAndRefresh()

```
void DisplayInterface::setLineAlignmentAndRefresh (
            uint8_t line,
            T_DisplayInterface_LineAlignment alignment )
```

Text alignment function.

This function updates the text alignment on the requested line. It calls the private function setLineAlignment to update data structure and then refreshes the display. Nothing is done if the requested alignment is the same than the current one, if the line is empty or if the line is in line shift mode.

Parameters

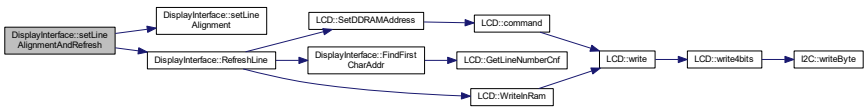
in	*line*	Requested line to update
in	*alignment*	Requested alignment for the text

Returns

Nothing

Definition at line 448 of file DisplayInterface.cpp.

Here is the call graph for this function:



3.7.3.11 shiftLine_task()

```
void DisplayInterface::shiftLine_task ( )  [static]
```

Line shifting periodic task.

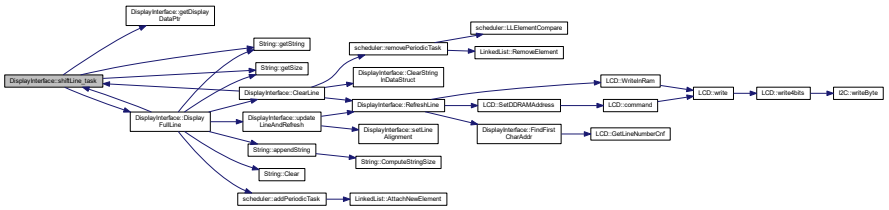
This function is called periodically by the scheduler. It shifts all the lines in line shifting mode and updates the data structures.

Returns

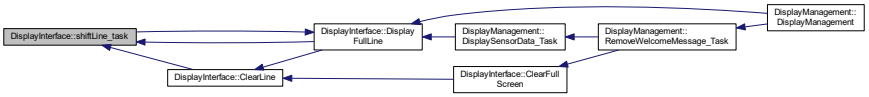
Nothing

Definition at line 282 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.3.12 updateLineAndRefresh()

```
void DisplayInterface::updateLineAndRefresh (
            uint8_t * str,
            uint8_t size,
            uint8_t line )  [private]
```

Line data string update function.

This function updates the data string and refreshes the display. The string is aligned according to the given parameter.

Parameters

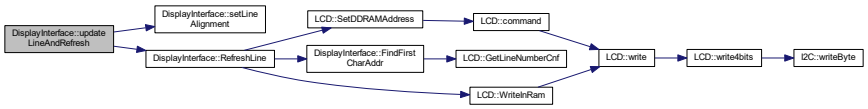
in	*str*	Pointer to the string to display
in	*size*	Size of the string
in	*line*	Line to update

Returns

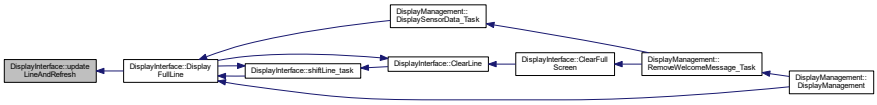
Nothing

Definition at line 145 of file DisplayInterface.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 display_data

```
T_display_data DisplayInterface::display_data[LCD_SIZE_NB_LINES]  [private]
```

Screen display data

Definition at line 165 of file DisplayInterface.h.

3.7.4.2 dummy

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

Needed for data alignment

Definition at line 164 of file DisplayInterface.h.

3.7.4.3 isShiftInProgress

```
bool DisplayInterface::isShiftInProgress  [private]
```

Flag indicating if a shift is in progress on any line

Definition at line 166 of file DisplayInterface.h.

3.7.4.4 p_lcd

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

Pointer to the attached LCD driver object

Definition at line 163 of file DisplayInterface.h.

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

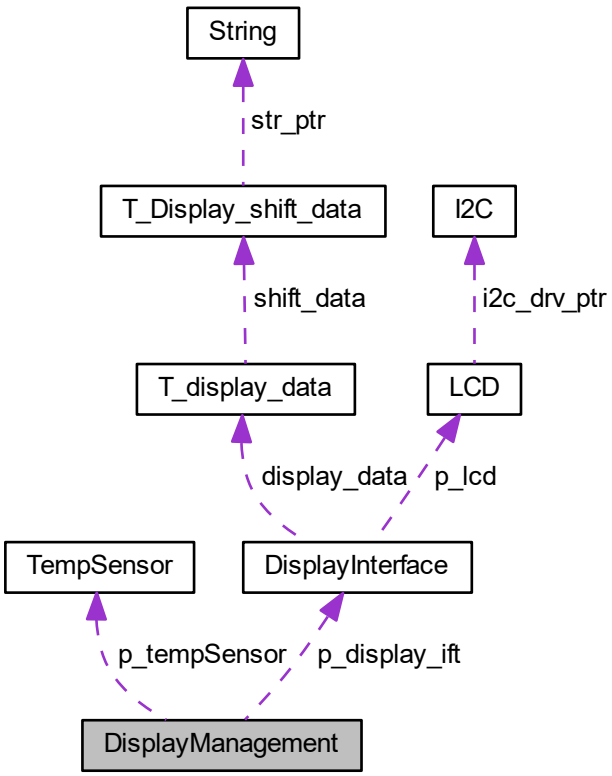
- DisplayInterface.h
- DisplayInterface.cpp

3.8 DisplayManagement Class Reference

Display management class.

```
#include <DisplayManagement.h>
```

Collaboration diagram for DisplayManagement:



Public Member Functions

- DisplayManagement ()

  Class constructor.
- DisplayInterface * GetIfPointer ()

  Interface pointer get function.
- TempSensor * GetTempSensorPtr ()

  Sensor pointer get function.

Static Public Member Functions

- static void DisplaySensorData_Task ()

  Periodic task for displaying sensor data.
- static void RemoveWelcomeMessage_Task ()

  End of welcome message task.

Private Attributes

- DisplayInterface * p_display_ift
- TempSensor * p_tempSensor

3.8.1 Detailed Description

Display management class.

This class manages all displays. It is a top-level class. It retrieves the data computed by other ASW classes and displays them. It is interfaced with DisplayInterface class to display data on screens. One interface class is used for each screen.

Definition at line 53 of file DisplayManagement.h.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 DisplayManagement()

```
DisplayManagement::DisplayManagement ( )
```

Class constructor.

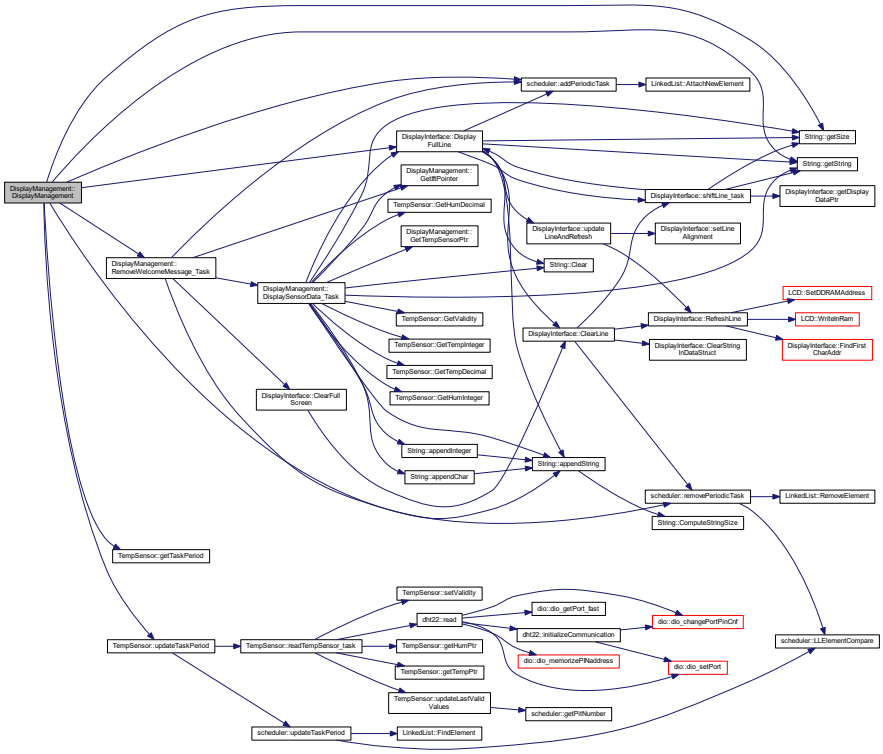
This class initializes display management.
It created a display interface object and initializes all class variables.

Returns

Nothing

Definition at line 30 of file DisplayManagement.cpp.

Here is the call graph for this function:



3.8.3 Member Function Documentation

3.8.3.1 DisplaySensorData_Task()

```
void DisplayManagement::DisplaySensorData_Task ( )  [static]
```

Periodic task for displaying sensor data.

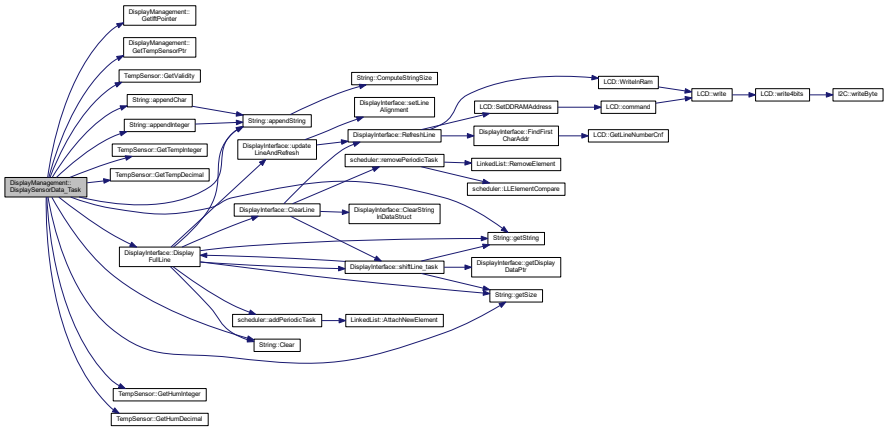
This function displays the sensors data on the screen. Currently temperature and humidity data coming from dht22 sensor are displayed.
It is called periodically by scheduler.

Returns

Nothing

Definition at line 74 of file DisplayManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.8.3.2 GetIftPointer()

```
DisplayInterface* DisplayManagement::GetIftPointer ( )  [inline]
```

Interface pointer get function.

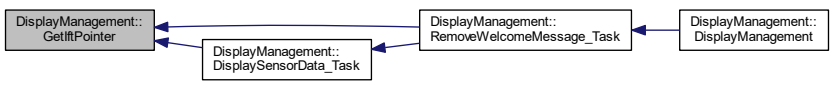
This function returns the pointer to the display interface object

Returns

Pointer to display interface object

Definition at line 82 of file DisplayManagement.h.

Here is the caller graph for this function:



3.8.3.3 GetTempSensorPtr()

```
TempSensor* DisplayManagement::GetTempSensorPtr ( )  [inline]
```

Sensor pointer get function.

This function returns the pointer to the temperature sensor object

Returns

Pointer to sensor object

Definition at line 93 of file DisplayManagement.h.

Here is the caller graph for this function:

3.8.3.4 RemoveWelcomeMessage_Task()

```
void DisplayManagement::RemoveWelcomeMessage_Task ( )  [static]
```

End of welcome message task.

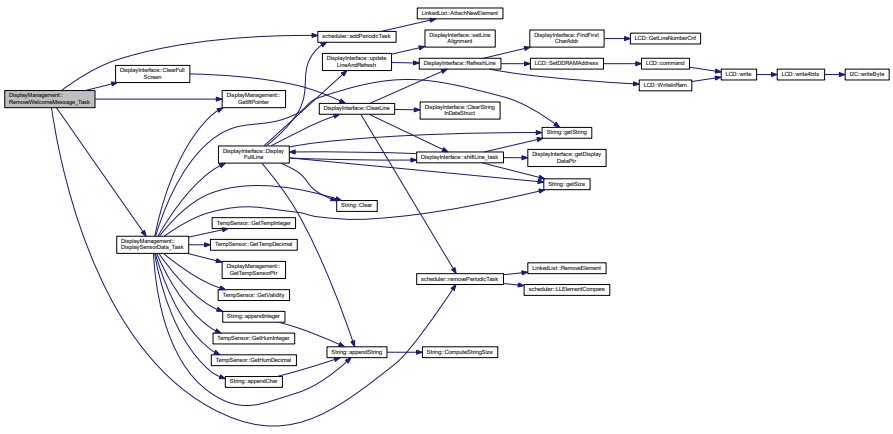
This task clears the welcome message from the screen and start periodic display of sensor data. This task shall be added in scheduler when the welcome message is displayed on screen. As it shall be called only once, the task removes itself from the scheduler after the first call.

Returns

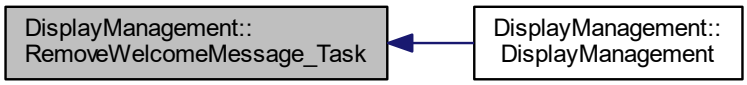
Nothing

Definition at line 57 of file DisplayManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.8.4 Member Data Documentation

3.8.4.1 p_display_ift

```
DisplayInterface* DisplayManagement::p_display_ift  [private]
```

Pointer to the display interface object

Definition at line 110 of file DisplayManagement.h.

3.8.4.2 p_tempSensor

```
TempSensor* DisplayManagement::p_tempSensor  [private]
```

Pointer to the temperature sensor object

Definition at line 111 of file DisplayManagement.h.

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

- DisplayManagement.h
- DisplayManagement.cpp

3.9 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

3.9.1 Detailed Description

Two-wire serial interface (I2C) class definition.

This class manages I2C driver.

Definition at line 23 of file I2C.h.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 I2C()

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

I2C class constructor.

This function initializes the I2C class and calls bus initialization function

Parameters

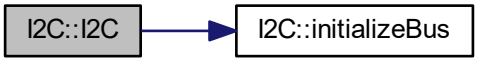
in	*l_bitrate*	Requested bitrate for I2C bus (in Hz)

Returns

Nothing

Definition at line 16 of file I2C.cpp.

Here is the call graph for this function:



3.9.3 Member Function Documentation

3.9.3.1 initializeBus()

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

I2C bus initialization.

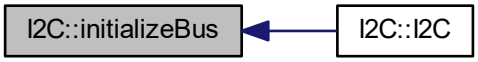
This function initializes the I2C bus, it resets the bus and configure the bitrate as requested. Bitrate is configured according to formula in the ATMEGA2560 datasheet : SCL freq = F_CPU / (16 + 2∗TWBR∗(4^TWPS)). Prescaler value is fixed to 1 (TWPS1 = 0 and TWPS0 = 0), then only TWBR value shall be computed.

Returns

Nothing

Definition at line 77 of file I2C.cpp.

Here is the caller graph for this function:



3.9.3.2 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 72 of file I2C.cpp.

3.9.3.3 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

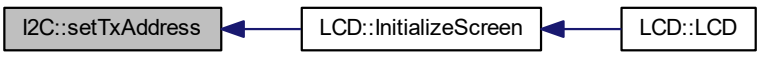
in	*address*	Requested Tx address

Returns

Nothing

Definition at line 67 of file I2C.cpp.

Here is the caller graph for this function:



3.9.3.4 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 24 of file I2C.cpp.

Here is the caller graph for this function:

3.9.4 Member Data Documentation

3.9.4.1 bitrate

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

Definition at line 63 of file I2C.h.

3.9.4.2 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

3.10 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.

3.10.1 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.

3.10.2 Constructor & Destructor Documentation

3.10.2.1 keepAliveLed()

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

Class constructor.

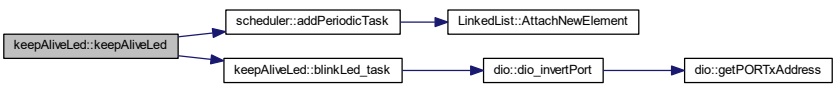
This function initializes the class keppAliveLed

Returns

Nothing

Definition at line 22 of file keepAliveLed.cpp.

Here is the call graph for this function:



3.10.3 Member Function Documentation

3.10.3.1 blinkLed_task()

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

Task for LED blinking.

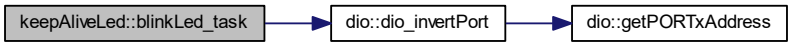
This function is inserted into the scheduler. It changes the state of the LED output to make it blink

Returns

Nothing

Definition at line 28 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:

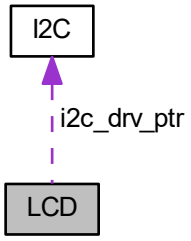
- keepAliveLed.h
- keepAliveLed.cpp

3.11 LCD Class Reference

Class for LCD S2004A display driver.

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

Collaboration diagram for LCD:



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 ConfigureI2CAddr (uint8_t param)

  I2C address 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 cnfI2C_addr
- I2C * i2c_drv_ptr
- uint8_t ddram_addr

3.11.1 Detailed Description

Class for LCD S2004A display driver.

This class handles functions managing LCD display S2004a on I2C bus

Definition at line 147 of file LCD.h.

3.11.2 Constructor & Destructor Documentation

3.11.2.1 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. It also creates a new instance of the I2C driver if needed.

Parameters

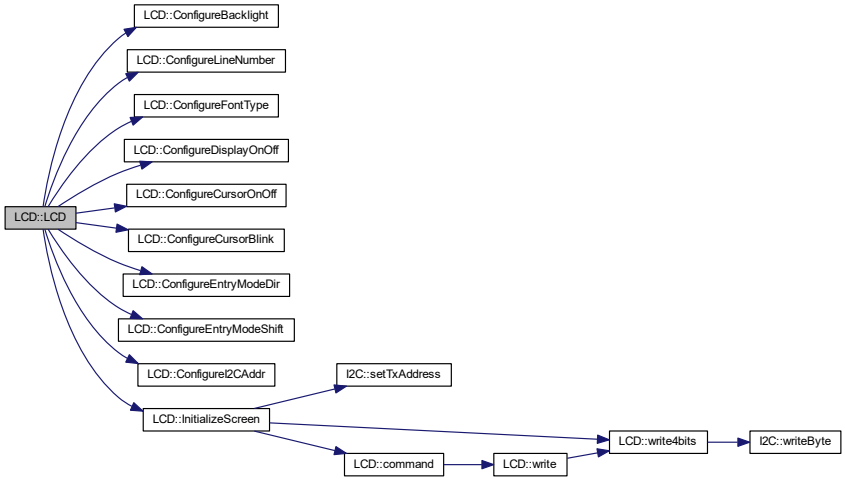
in	*init_conf*	Initial configuration structure

Returns

Nothing

Definition at line 18 of file LCD.cpp.

Here is the call graph for this function:



3.11.3 Member Function Documentation

3.11.3.1 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

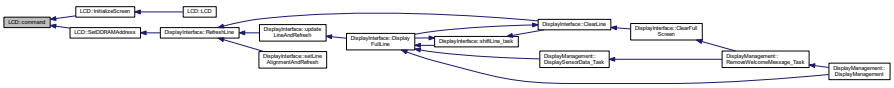
Returns

Nothing

Definition at line 125 of file LCD.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



3.11.3.2 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 178 of file LCD.h.

Here is the caller graph for this function:

3.11.3.3 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 238 of file LCD.h.

Here is the caller graph for this function:

3.11.3.4 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 226 of file LCD.h.

Here is the caller graph for this function:

3.11.3.5 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 214 of file LCD.h.

Here is the caller graph for this function:

3.11.3.6 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 250 of file LCD.h.

Here is the caller graph for this function:

3.11.3.7 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 262 of file LCD.h.

Here is the caller graph for this function:

3.11.3.8 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

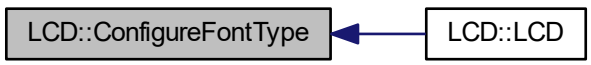
in	*param*	Configuration value

Returns

Nothing

Definition at line 202 of file LCD.h.

Here is the caller graph for this function:



3.11.3.9 ConfigureI2CAddr()

```
void LCD::ConfigureI2CAddr (
            uint8_t param )  [inline]
```

I2C address configuration function.

This function configures the I2V address of the LCD screen according to the parameter.

Parameters

in	*param*	I2C address

Returns

Nothing

Definition at line 274 of file LCD.h.

Here is the caller graph for this function:

3.11.3.10 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 190 of file LCD.h.

Here is the caller graph for this function:

3.11.3.11 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 294 of file LCD.h.

3.11.3.12 GetLineNumberCnf()

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

Number of line get function.

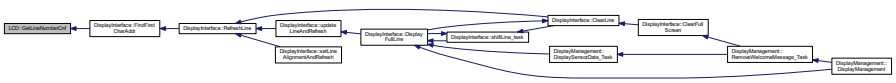
This function returns the line number configuration of the screen : 1 or 2 lines mode.

Returns

Line number configuration

Definition at line 316 of file LCD.h.

Here is the caller graph for this function:



3.11.3.13 InitializeScreen()

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

Screen configuration function.

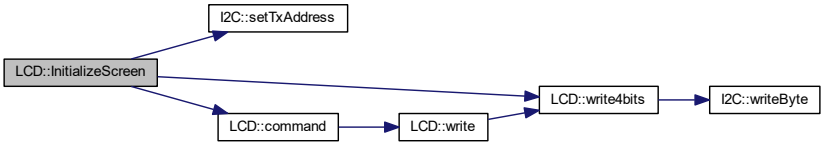
This function configures the LCD screen. It's must be called during initialization phase, or the screen won't be usable. The configuration process is described in LCD datasheet J2004A-GFDN-DYNC

Returns

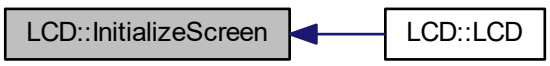
Nothing

Definition at line 73 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.11.3.14 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

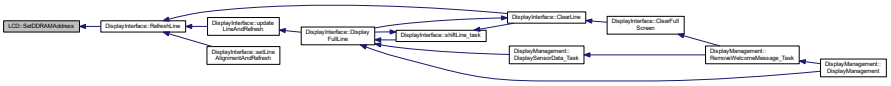
Returns

Nothing

Definition at line 168 of file LCD.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



3.11.3.15 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

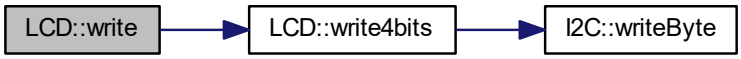
in	*data*	8-bit data for D0-7 pins of screen
in	*mode*	Requested mode for LCD communication

Returns

Nothing

Definition at line 62 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.11.3.16 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

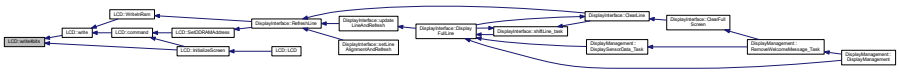
Returns

Nothing

Definition at line 45 of file LCD.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



3.11.3.17 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 190 of file LCD.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

3.11.4 Member Data Documentation

3.11.4.1 backlight_enable

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

Backlight enable flag

Definition at line 324 of file LCD.h.

3.11.4.2 cnfCursorBlink

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

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

Definition at line 329 of file LCD.h.

3.11.4.3 cnfCursorOnOff

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

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

Definition at line 328 of file LCD.h.

3.11.4.4 cnfDisplayOnOff

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

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

Definition at line 327 of file LCD.h.

3.11.4.5 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 330 of file LCD.h.

3.11.4.6 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 331 of file LCD.h.

3.11.4.7 cnfFontType

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

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

Definition at line 326 of file LCD.h.

3.11.4.8 cnfI2C_addr

```
uint8_t LCD::cnfI2C_addr  [private]
```

I2C address of the LCD screen

Definition at line 332 of file LCD.h.

3.11.4.9 cnfLineNumber

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

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

Definition at line 325 of file LCD.h.

3.11.4.10 ddram_addr

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

Screen DDRAM address

Definition at line 336 of file LCD.h.

3.11.4.11 i2c_drv_ptr

```
I2C* LCD::i2c_drv_ptr  [private]
```

Pointer to the I2C driver object

Definition at line 334 of file LCD.h.

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

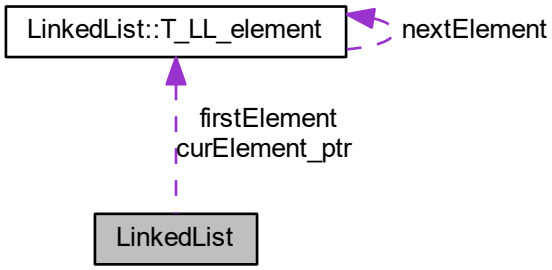
- LCD.h
- LCD.cpp

3.12 LinkedList Class Reference

Linked list class.

```
#include <LinkedList.h>
```

Collaboration diagram for LinkedList:



Classes

- struct T_LL_element

  Type defining a linked list element.

Public Member Functions

- LinkedList ()

  Class constructor.
- ~LinkedList ()

  Class destructor.
- void AttachNewElement (void *data_ptr)

  Add an new element to the list.
- bool RemoveElement (CompareFctPtr_t comparisonFct_ptr, void *reference_ptr)

  Removes an element from the chain.
- void * getCurrentElement ()

  Current element get function.
- bool MoveToNextElement ()

  Move to next element function.
- void ResetElementPtr ()

  Resets element pointer.
- bool IsLLEmpty ()

  Empty linked list.
- bool FindElement (CompareFctPtr_t comparisonFct_ptr, void *reference_ptr, void **chainElement_ptr)

  Element finding function.

Private Types

- typedef struct LinkedList::T_LL_element T_LL_element

  Type defining a linked list element.

Private Attributes

- T_LL_element * firstElement
- T_LL_element * curElement_ptr

3.12.1 Detailed Description

Linked list class.

This class defines a linked list and the associated services.
All classes using a linked list with this interface shall implement a comparison function used to find the list element to remove. This function shall have the following prototype : static bool LLElementCompare(void* LLElement, void* CompareElement);

Definition at line 22 of file LinkedList.h.

3.12.2 Member Typedef Documentation

3.12.2.1 T_LL_element

```
typedef struct LinkedList::T_LL_element LinkedList::T_LL_element  [private]
```

Type defining a linked list element.

This structure defines a linked list element. An element is defined by a pointer to the attached data and a pointer to the next element.

3.12.3 Constructor & Destructor Documentation

3.12.3.1 LinkedList()

```
LinkedList::LinkedList ( )
```

Class constructor.

This constructor initializes a linked list

Returns

Nothing

Definition at line 37 of file LinkedList.cpp.

3.12.3.2 ~LinkedList()

```
LinkedList::~LinkedList ( )
```

Class destructor.

This function deletes the linked list

Returns

Nothing

Definition at line 43 of file LinkedList.cpp.

Here is the call graph for this function:

3.12.4 Member Function Documentation

3.12.4.1 AttachNewElement()

```
void LinkedList::AttachNewElement (
            void * data_ptr )
```

Add an new element to the list.

This function adds a new element at the end of the list. The data pointer to attach to the element is given in parameter

Parameters

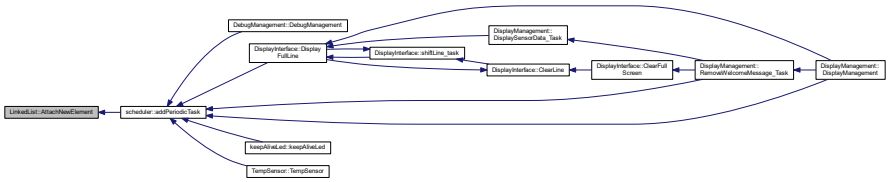
in	*data_ptr*	Pointer to the data element

Returns

Nothing

Definition at line 61 of file LinkedList.cpp.

Here is the caller graph for this function:



3.12.4.2 FindElement()

```
bool LinkedList::FindElement (
            CompareFctPtr_t comparisonFct_ptr,
            void * reference_ptr,
            void ** chainElement_ptr )
```

Element finding function.

This function finds the given element reference_ptr inside the chain. The comparison between the elements of the chain and the reference element is done using the given comparison function.

Parameters

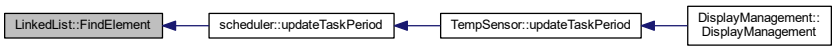
in	*comparisonFct_ptr*	Pointer to the comparison function
in	*reference_ptr*	Pointer to the element to find in the chain
out	*chainElement_ptr*	Pointer to pointer to the found element

Returns

True if the element has been found in the chain, false otherwise

Definition at line 152 of file LinkedList.cpp.

Here is the caller graph for this function:



3.12.4.3 getCurrentElement()

```
void* LinkedList::getCurrentElement ( )  [inline]
```

Current element get function.

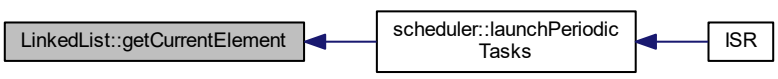
This function returns a pointer to the current pointed data in the chain.

Returns

Pointer to the current data

Definition at line 67 of file LinkedList.h.

Here is the caller graph for this function:



3.12.4.4 IsLLEmpty()

```
bool LinkedList::IsLLEmpty ( )
```

Empty linked list.

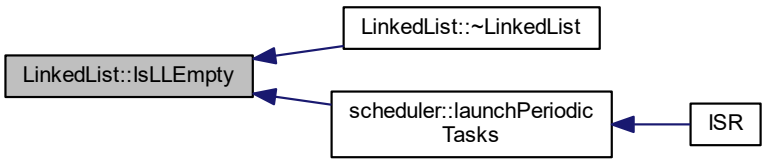
This function checks whether the linked list is empty or not (pointer to first element is equal to 0 or not).

Returns

True if the list is empty, false otherwise

Definition at line 144 of file LinkedList.cpp.

Here is the caller graph for this function:



3.12.4.5 MoveToNextElement()

```
bool LinkedList::MoveToNextElement ( )
```

Move to next element function.

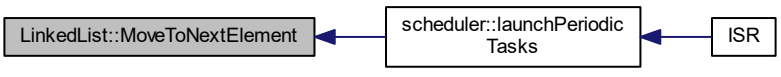
This function moves the element pointer to the next element of the chain.

Returns

True if the next element exists, false if there is no next element

Definition at line 130 of file LinkedList.cpp.

Here is the caller graph for this function:



3.12.4.6 RemoveElement()

```
bool LinkedList::RemoveElement (
            CompareFctPtr_t comparisonFct_ptr,
            void * reference_ptr )
```

Removes an element from the chain.

This function removes an element from the chain. To know which element shall be removed, we use the comparison function given in parameter. This function is called with two parameters : the data pointer from the chain and the reference pointer given as parameter.

Parameters

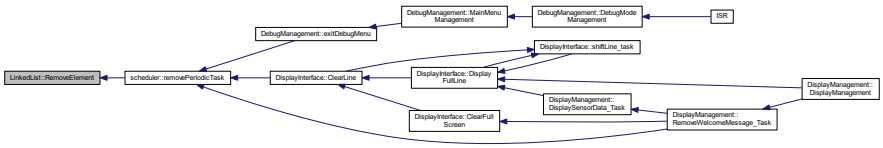
in	*comparisonFct_ptr*	Pointer to the comparison function to use
in	*reference_ptr*	Pointer to the reference data used for comparison

Returns

True if the element has been correctly removed from the chain, false otherwise

Definition at line 86 of file LinkedList.cpp.

Here is the caller graph for this function:



3.12.4.7 ResetElementPtr()

```
void LinkedList::ResetElementPtr ( )  [inline]
```

Resets element pointer.

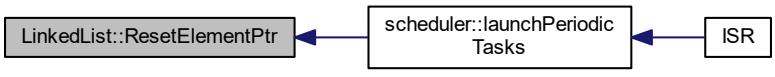
This function sets the element pointer to the first element of the chain.

Returns

Nothing

Definition at line 86 of file LinkedList.h.

Here is the caller graph for this function:



3.12.5 Member Data Documentation

3.12.5.1 curElement_ptr

```
T_LL_element* LinkedList::curElement_ptr  [private]
```

Pointer to the current element of the list

Definition at line 125 of file LinkedList.h.

3.12.5.2 firstElement

```
T_LL_element* LinkedList::firstElement  [private]
```

Pointer to the first element of the list

Definition at line 124 of file LinkedList.h.

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

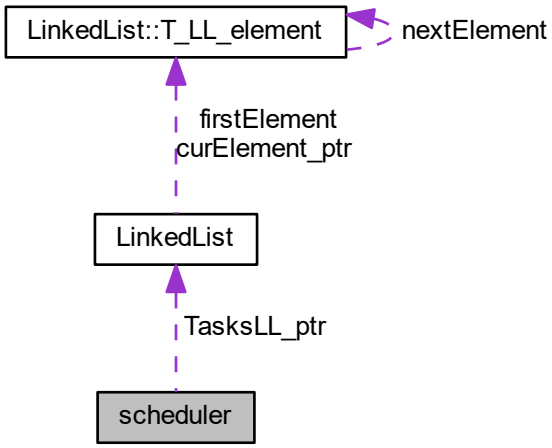
- LinkedList.h
- LinkedList.cpp

3.13 scheduler Class Reference

Scheduler class.

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

Collaboration diagram for scheduler:



Classes

- 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.
- bool updateTaskPeriod (TaskPtr_t task_ptr, uint16_t period)

  Task period update function.
- uint8_t getTaskCount ()

  Task count get function.

Static Public Member Functions

- static bool LLElementCompare (void ∗LLElement, void ∗CompareElement)

  Linked list comparison function.

Private Types

- typedef struct scheduler::Task_t Task_t

  Type defining a task structure.

Private Attributes

- uint8_t task_count
- LinkedList ∗ TasksLL_ptr
- uint32_t pit_number

3.13.1 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.
All tasks called by the scheduler shall have the following prototype : static void task();

Definition at line 30 of file scheduler.h.

3.13.2 Member Typedef Documentation

3.13.2.1 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) and a period.

3.13.3 Constructor & Destructor Documentation

3.13.3.1 scheduler()

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

scheduler class constructor

This function initializes the class scheduler

Returns

Nothing

Definition at line 29 of file scheduler.cpp.

Here is the call graph for this function:

3.13.4 Member Function Documentation

3.13.4.1 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

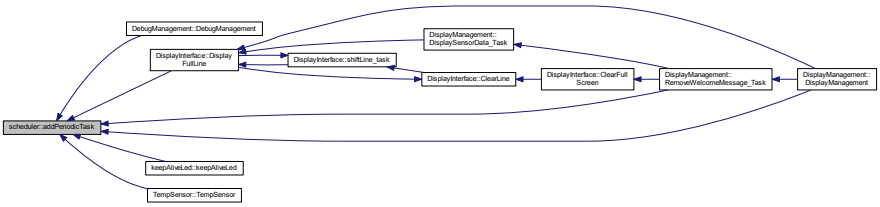
Returns

Nothing

Definition at line 99 of file scheduler.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



3.13.4.2 getPitNumber()

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

Get function for PIT number.

This function returns the PIT number

Returns

PIT number

Definition at line 113 of file scheduler.cpp.

Here is the caller graph for this function:

3.13.4.3 getTaskCount()

```
uint8_t scheduler::getTaskCount ( )  [inline]
```

Task count get function.

This function returns the current number of tasks managed by scheduler.

Returns

Number of tasks

Definition at line 115 of file scheduler.h.

3.13.4.4 launchPeriodicTasks()

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

Main scheduler function.

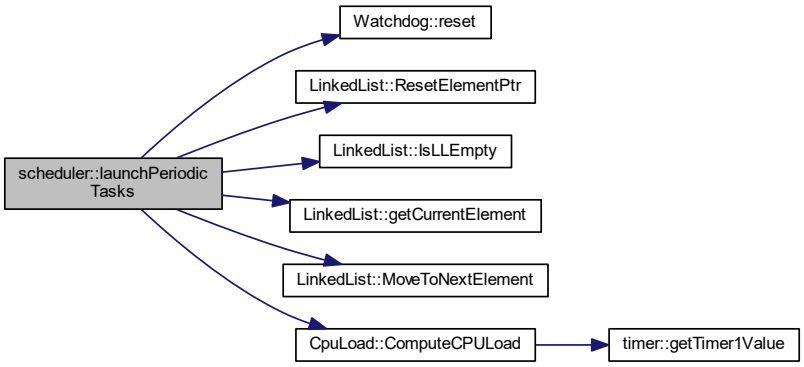
This function launches the scheduled tasks according to current software time and task configuration

Returns

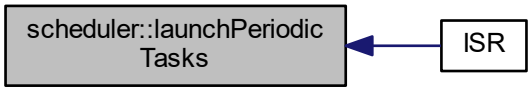
Nothing

Definition at line 54 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.13.4.5 LLElementCompare()

```
bool scheduler::LLElementCompare (
            void * LLElement,
            void * CompareElement )  [static]
```

Linked list comparison function.

This function is called by the linked list class to compare one element of the list to a given element. In the class scheduler, the LLElement is a task pointer (containing a function pointer and a period), and the compareElement a function pointer. The comparison will be done between the two function pointer.

Parameters

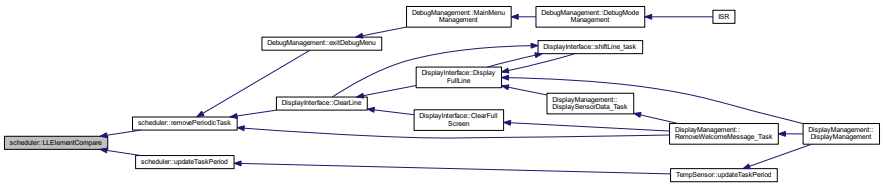
in	*LLElement*	Pointer to the linked list element
in	*CompareElement*	Pointer to the element to the compare

Returns

True if both elements are identical, false otherwise

Definition at line 131 of file scheduler.cpp.

Here is the caller graph for this function:



3.13.4.6 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

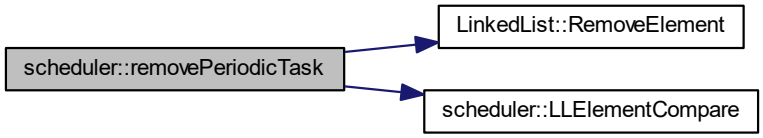
in	*task_ptr*	address of the task to remove from scheduler

Returns

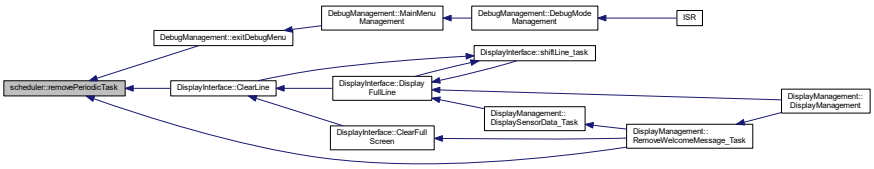
TRUE if the task has been removed, FALSE if the task does not exist in the scheduler

Definition at line 119 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.13.4.7 startScheduling()

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

Starts the tasks scheduling.

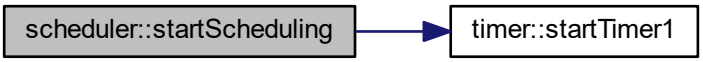
This function starts the timer which will trigger an interrupt every software period. When the interrupt is raised the scheduler will launch applications

Returns

Nothing

Definition at line 93 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.13.4.8 updateTaskPeriod()

```
bool scheduler::updateTaskPeriod (
            TaskPtr_t task_ptr,
            uint16_t period )
```

Task period update function.

This function updates the period of the given task. The task is never stopped during the process, only the period value is updated.

Parameters

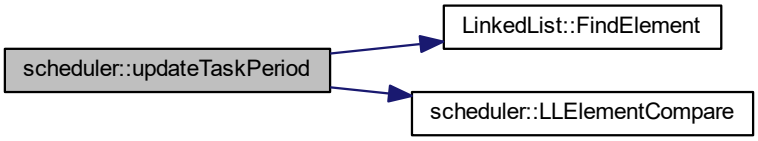
in	*task_ptr*	Pointer of the task to update
in	*period*	New period of the task

Returns

True if the update has been correctly done, false otherwise

Definition at line 142 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

3.13.5 Member Data Documentation

3.13.5.1 pit_number

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

Counter of periodic interrupts

Definition at line 140 of file scheduler.h.

3.13.5.2 task_count

```
uint8_t scheduler::task_count  [private]
```

Number of task in scheduler

Definition at line 136 of file scheduler.h.

3.13.5.3 TasksLL_ptr

```
LinkedList* scheduler::TasksLL_ptr  [private]
```

Pointer to the linked list object containing the tasks

Definition at line 138 of file scheduler.h.

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

- scheduler.h
- scheduler.cpp

3.14 String Class Reference

String management class.

`#include <String.h>`

Public Member Functions

- String (const uint8_t *str)
  Class constructor.
- String ()
  Class constructor.
- ~String ()
  Class destructor.
- uint8_t * getString ()
  String pointer get function.
- uint8_t getSize ()
  Size get function.
- void appendString (uint8_t *str)
  String adding function.
- void appendInteger (uint16_t value, uint8_t base)
  Integer adding function.
- void appendBool (bool data, bool isText)
  Boolean adding function.
- void appendChar (uint8_t data)
  Character adding function.
- void Clear ()
  String clear function.

Private Member Functions

- uint8_t ComputeStringSize (uint8_t *str)
  String size computation function.

Private Attributes

- uint8_t * string
- uint8_t size

3.14.1 Detailed Description

String management class.

This class defines string object. It implements some functions to manage chains of characters. The string is limited to 255 characters. It must finish by the character '\0'.

Definition at line 18 of file String.h.

3.14.2 Constructor & Destructor Documentation

3.14.2.1 String() [1/2]

```
String::String (
            const uint8_t * str )
```

Class constructor.

This function initializes the class. The string is initialized with the data given in parameter.

Parameters

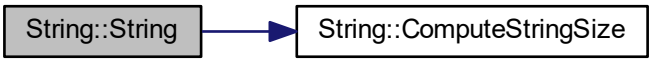
in	*str*	Pointer to initialization string

Returns

Nothing

Definition at line 15 of file String.cpp.

Here is the call graph for this function:



3.14.2.2 String() [2/2]

```
String::String ( )
```

Class constructor.

This function initializes the class with an empty string. The size is set to 0.

Returns

Nothing

Definition at line 33 of file String.cpp.

3.14.2.3 ~String()

```
String::~String ( )
```

Class destructor.

This function frees the memory used to contain the string when the object is deleted

Returns

Nothing

Definition at line 39 of file String.cpp.

Here is the call graph for this function:

3.14.3 Member Function Documentation

3.14.3.1 appendBool()

```
void String::appendBool (
            bool data,
            bool isText )
```

Boolean adding function.

This functions adds the given boolean data at the end of the main string. The string size is updated accordingly. According to the input parameter isText, the boolean parameter is converted into a string (true/false) or an integer (0/1).

Parameters

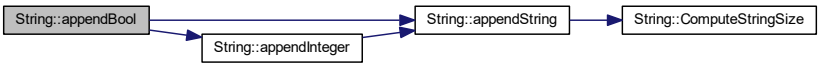
in	*data*	Boolean data to add
in	*isText*	Defines the conversion mode : text or integer

Returns

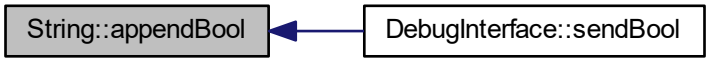
Nothing

Definition at line 121 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.14.3.2 appendChar()

```
void String::appendChar (
            uint8_t data )
```

Character adding function.

This functions adds the given character at the end of the main string. The string size is updated by 1.

Parameters

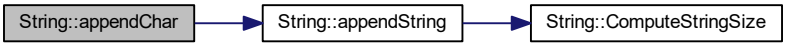
in	*data*	1-byte character to add

Returns

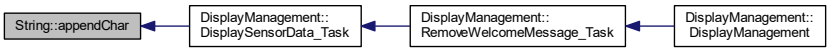
Nothing

Definition at line 139 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.14.3.3 appendInteger()

```
void String::appendInteger (
            uint16_t value,
            uint8_t base )
```

Integer adding function.

This functions adds the given integer at the end of the main string. The string size is updated accordingly. The integer parameter is first converted into a chain of character according to the base and then added to the string.

Parameters

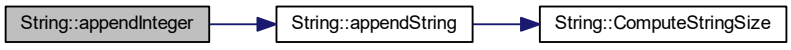
in	*value*	Integer to add
in	*base*	Base of computation of the integer (between 2 and 36)

Returns

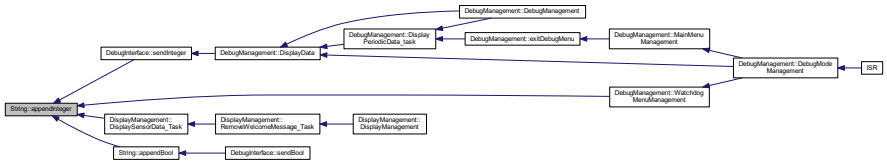
Nothing

Definition at line 95 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.14.3.4 appendString()

```
void String::appendString (
            uint8_t * str )
```

String adding function.

This functions adds the given string at the end of the main string. The string size is updated accordingly.

Parameters

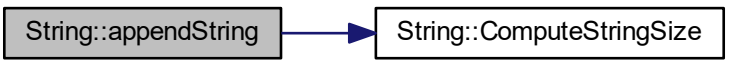
in	*str*	New string to add

Returns

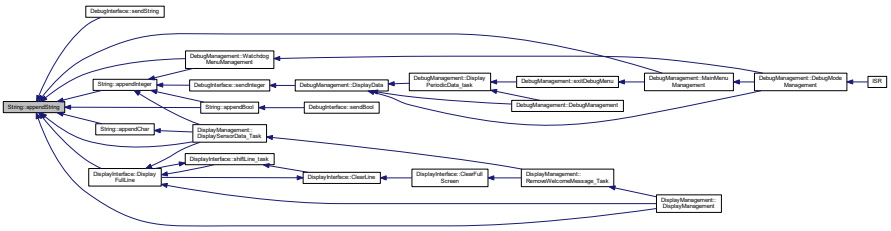
Nothing

Definition at line 57 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.14.3.5 Clear()

```
void String::Clear ( )
```

String clear function.

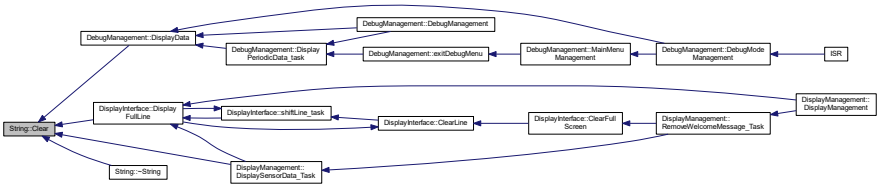
This function clears the string. Size is set to 0 and the memory is freed.

Returns

Nothing

Definition at line 113 of file String.cpp.

Here is the caller graph for this function:



3.14.3.6 ComputeStringSize()

```
uint8_t String::ComputeStringSize (
            uint8_t * str )  [private]
```

String size computation function.

This function computes the sizes of the given string. It counts the number of character between the start of the string given in parameter and the next \0 character.

Parameters

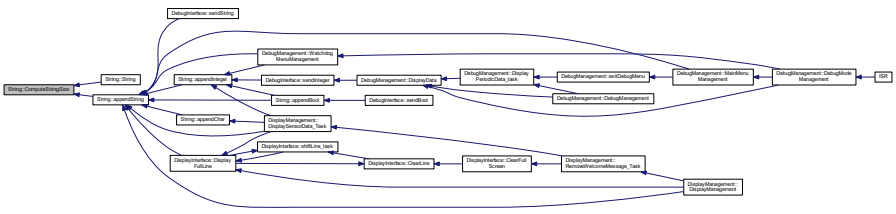
in	*str*	Pointer to the beginning of the string

Returns

Number of character of the string (the \0 is excluded)

Definition at line 44 of file String.cpp.

Here is the caller graph for this function:



3.14.3.7 getSize()

```
uint8_t String::getSize ( )  [inline]
```

Size get function.

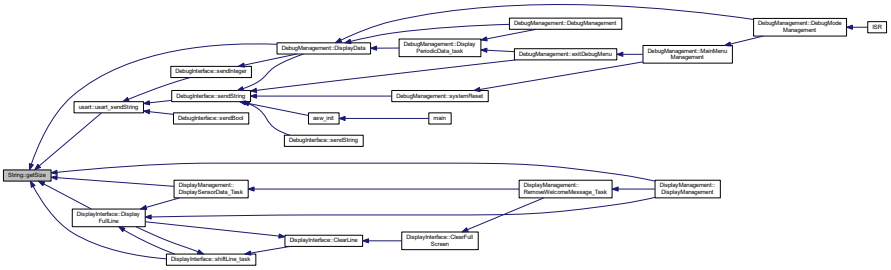
This function returns the size of the string.

Returns

Size of the string

Definition at line 64 of file String.h.

Here is the caller graph for this function:



3.14.3.8 getString()

```
uint8_t* String::getString ( )  [inline]
```

String pointer get function.

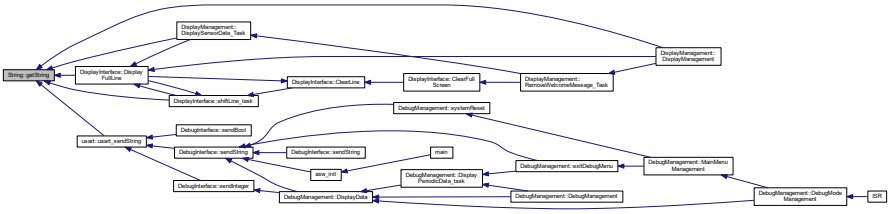
This function returns the pointer to the beginning of the string.

Returns

String pointer

Definition at line 53 of file String.h.

Here is the caller graph for this function:



3.14.4 Member Data Documentation

3.14.4.1 size

```
uint8_t String::size  [private]
```

Size of the string (the '\0' at the end of the string is not taken into account

Definition at line 121 of file String.h.

3.14.4.2 string

```
uint8_t* String::string  [private]
```

Pointer to the start of the string

Definition at line 120 of file String.h.

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

- String.h
- String.cpp

3.15 T_ASW_init_cnf Struct Reference

ASW initialization configuration structure.

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

Public Attributes

- bool isDebugActivated
- bool isLEDActivated
- bool isTempSensorActivated
- bool isDisplayActivated

3.15.1 Detailed Description

ASW initialization configuration structure.

This structure is used to define which ASW services shall be started at SW start-up.

Definition at line 17 of file asw.h.

3.15.2 Member Data Documentation

3.15.2.1 isDebugActivated

```
bool T_ASW_init_cnf::isDebugActivated
```

Debug services activation flag

Definition at line 19 of file asw.h.

3.15.2.2 isDisplayActivated

```
bool T_ASW_init_cnf::isDisplayActivated
```

LCD display activation flag

Definition at line 22 of file asw.h.

3.15.2.3 isLEDActivated

```
bool T_ASW_init_cnf::isLEDActivated
```

Keep-alive LED activation flag

Definition at line 20 of file asw.h.

3.15.2.4 isTempSensorActivated

```
bool T_ASW_init_cnf::isTempSensorActivated
```

Temperature sensor activation flag

Definition at line 21 of file asw.h.

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

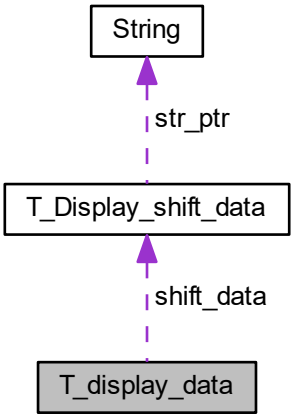
- asw.h

3.16 T_display_data Struct Reference

Structure containing display data.

`#include <DisplayInterface.h>`

Collaboration diagram for T_display_data:



Public Attributes

- bool isEmpty
- T_DisplayInterface_LineDisplayMode mode
- T_DisplayInterface_LineAlignment alignment
- T_Display_shift_data shift_data
- uint8_t display_str [LCD_SIZE_NB_CHAR_PER_LINE]

3.16.1 Detailed Description

Structure containing display data.

This structure contains all data used for screen display

Definition at line 57 of file DisplayInterface.h.

3.16.2 Member Data Documentation

3.16.2.1 alignment

```
T_DisplayInterface_LineAlignment T_display_data::alignment
```

Line alignment

Definition at line 61 of file DisplayInterface.h.

3.16.2.2 display_str

```
uint8_t T_display_data::display_str[LCD_SIZE_NB_CHAR_PER_LINE]
```

Current string displayed on the screen

Definition at line 63 of file DisplayInterface.h.

3.16.2.3 isEmpty

```
bool T_display_data::isEmpty
```

Flag indicating if the line is empty or not

Definition at line 59 of file DisplayInterface.h.

3.16.2.4 mode

```
T_DisplayInterface_LineDisplayMode T_display_data::mode
```

Current display mode

Definition at line 60 of file DisplayInterface.h.

3.16.2.5 shift_data

```
T_Display_shift_data T_display_data::shift_data
```

Shift data for the current line

Definition at line 62 of file DisplayInterface.h.

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

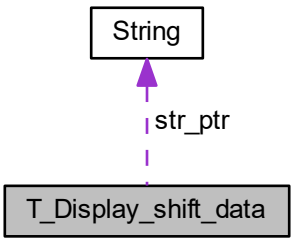
- DisplayInterface.h

3.17 T_Display_shift_data Struct Reference

Structure containing shift data.

`#include <DisplayInterface.h>`

Collaboration diagram for T_Display_shift_data:



Public Attributes

- String * str_ptr
- uint8_t * str_cur_ptr
- uint8_t temporization

3.17.1 Detailed Description

Structure containing shift data.

This structure contains all useful data for line shifting. These data need to be kept between each call of the periodic function.

Definition at line 45 of file DisplayInterface.h.

3.17.2 Member Data Documentation

3.17.2.1 str_cur_ptr

`uint8_t* T_Display_shift_data::str_cur_ptr`

Pointer to the address of the first displayed character

Definition at line 48 of file DisplayInterface.h.

3.17.2.2 str_ptr

```
String* T_Display_shift_data::str_ptr
```

Pointer to the start address of the string

Definition at line 47 of file DisplayInterface.h.

3.17.2.3 temporization

```
uint8_t T_Display_shift_data::temporization
```

Shifting period

Definition at line 49 of file DisplayInterface.h.

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

- DisplayInterface.h

3.18 T_LCD_conf_struct Struct Reference

Structure defining LCD configuration.

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

Public Attributes

- uint32_t i2c_bitrate
- uint8_t i2c_addr
- bool backlight_en
- bool lineNumber_cnf
- bool fontType_cnf
- bool display_en
- bool cursor_en
- bool cursorBlink_en
- bool entryModeDir
- bool entryModeShift

3.18.1 Detailed Description

Structure defining LCD configuration.

Definition at line 128 of file LCD.h.

3.18.2 Member Data Documentation

3.18.2.1 backlight_en

```
bool T_LCD_conf_struct::backlight_en
```

Screen backlight enable flag

Definition at line 132 of file LCD.h.

3.18.2.2 cursor_en

```
bool T_LCD_conf_struct::cursor_en
```

Screen cursor enable flag

Definition at line 136 of file LCD.h.

3.18.2.3 cursorBlink_en

```
bool T_LCD_conf_struct::cursorBlink_en
```

Screen cursor blinking enable flag

Definition at line 137 of file LCD.h.

3.18.2.4 display_en

```
bool T_LCD_conf_struct::display_en
```

Screen display enable flag

Definition at line 135 of file LCD.h.

3.18.2.5 entryModeDir

```
bool T_LCD_conf_struct::entryModeDir
```

Entry mode direction configuration

Definition at line 138 of file LCD.h.

3.18.2.6 entryModeShift

```
bool T_LCD_conf_struct::entryModeShift
```

Entry mode shift configuration

Definition at line 139 of file LCD.h.

3.18.2.7 fontType_cnf

```
bool T_LCD_conf_struct::fontType_cnf
```

Font configuration

Definition at line 134 of file LCD.h.

3.18.2.8 i2c_addr

```
uint8_t T_LCD_conf_struct::i2c_addr
```

I2C address if the screen

Definition at line 131 of file LCD.h.

3.18.2.9 i2c_bitrate

```
uint32_t T_LCD_conf_struct::i2c_bitrate
```

I2C bitrate needed by the LCD screen

Definition at line 130 of file LCD.h.

3.18.2.10 lineNumber_cnf

```
bool T_LCD_conf_struct::lineNumber_cnf
```

Screen line number configuration (1 or 2 lines)

Definition at line 133 of file LCD.h.

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

- LCD.h

3.19 LinkedList::T_LL_element Struct Reference

Type defining a linked list element.

Collaboration diagram for LinkedList::T_LL_element:

Public Attributes

- void * data_ptr
- T_LL_element * nextElement

3.19.1 Detailed Description

Type defining a linked list element.

This structure defines a linked list element. An element is defined by a pointer to the attached data and a pointer to the next element.

Definition at line 117 of file LinkedList.h.

3.19.2 Member Data Documentation

3.19.2.1 data_ptr

```
void* LinkedList::T_LL_element::data_ptr
```

Definition at line 119 of file LinkedList.h.

3.19.2.2 nextElement

```
T_LL_element* LinkedList::T_LL_element::nextElement
```

Definition at line 120 of file LinkedList.h.

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

- LinkedList.h

3.20 scheduler::Task_t Struct Reference

Type defining a task structure.

Public Attributes

- TaskPtr_t TaskPtr
- uint16_t period

3.20.1 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) and a period.

Definition at line 129 of file scheduler.h.

3.20.2 Member Data Documentation

3.20.2.1 period

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

Period of the task

Definition at line 132 of file scheduler.h.

3.20.2.2 TaskPtr

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

Pointer to the task

Definition at line 131 of file scheduler.h.

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

- scheduler.h

3.21 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 ()
- uint8_t GetTempInteger ()
  Temperature formatting function - Integer part.
- uint8_t GetTempDecimal ()
  Temperature formatting function - Decimal part.
- uint8_t GetHumInteger ()
  Humidity formatting function - Integer part.
- uint8_t GetHumDecimal ()
  Humidity formatting function - Decimal part.
- bool GetValidity ()
  Data validity get function.
- bool updateTaskPeriod (uint16_t period)
  Task period update.
- uint16_t getTaskPeriod ()
  Task period get function.

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
- uint16_t task_period

3.21.1 Detailed Description

Class for temperature sensor.

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

Definition at line 21 of file TempSensor.h.

3.21.2 Constructor & Destructor Documentation

3.21.2.1 TempSensor()

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

Class constructor.

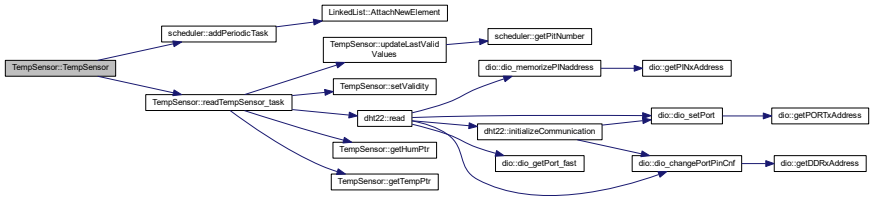
This function initializes all data of the class TempSensor. If needed, it creates a new instance of the DHT22 sensor object. It also adds the periodic task in the scheduler.

Returns

Nothing

Definition at line 24 of file TempSensor.cpp.

Here is the call graph for this function:



3.21.3 Member Function Documentation

3.21.3.1 GetHumDecimal()

```
uint8_t TempSensor::GetHumDecimal ( )  [inline]
```

Humidity formatting function - Decimal part.

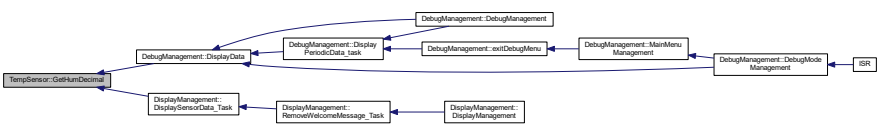
This function return the decimal part of the humidity

Returns

Decimal value of the humidity

Definition at line 124 of file TempSensor.h.

Here is the caller graph for this function:



3.21.3.2 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 80 of file TempSensor.cpp.

3.21.3.3 GetHumInteger()

```
uint8_t TempSensor::GetHumInteger ( )  [inline]
```

Humidity formatting function - Integer part.

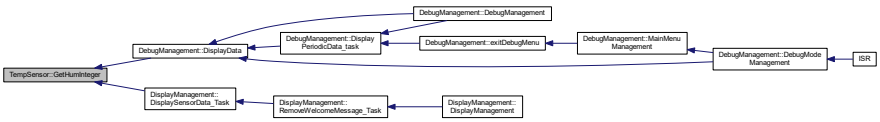
This function return the integer part of the humidity

Returns

Integer value of the humidity

Definition at line 113 of file TempSensor.h.

Here is the caller graph for this function:



3.21.3.4 getHumPtr()

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

Get pointer to data raw_humidity.

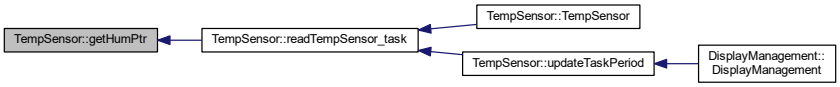
This function returns a pointer to the class member raw_humidity

Returns

Pointer to raw_humidity

Definition at line 55 of file TempSensor.cpp.

Here is the caller graph for this function:



3.21.3.5 getTaskPeriod()

```
uint16_t TempSensor::getTaskPeriod ( )  [inline]
```

Task period get function.

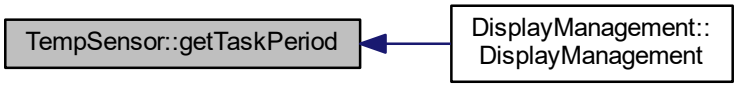
This function returns the period of the sensor task

Returns

Period of the task (ms)

Definition at line 155 of file TempSensor.h.

Here is the caller graph for this function:



3.21.3.6 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 86 of file TempSensor.cpp.

3.21.3.7 GetTempDecimal()

```
uint8_t TempSensor::GetTempDecimal ( )  [inline]
```

Temperature formatting function - Decimal part.

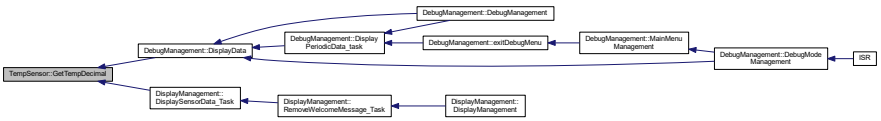
This function return the decimal part of the temperature

Returns

Decimal value of the temperature

Definition at line 102 of file TempSensor.h.

Here is the caller graph for this function:



3.21.3.8 GetTempInteger()

```
uint8_t TempSensor::GetTempInteger ( )  [inline]
```

Temperature formatting function - Integer part.

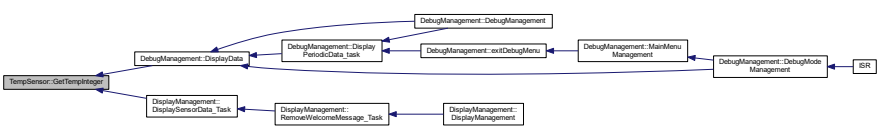
This function return the integer part of the temperature

Returns

Integer value of the temperature

Definition at line 91 of file TempSensor.h.

Here is the caller graph for this function:



3.21.3.9 getTempPtr()

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

Get pointer to data raw_temperature.

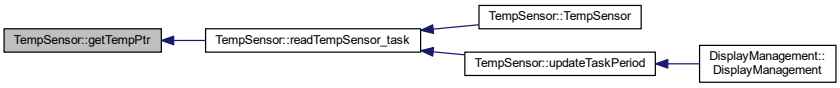
This function returns a pointer to the class member raw_temperature

Returns

Pointer to raw_temperature

Definition at line 60 of file TempSensor.cpp.

Here is the caller graph for this function:



3.21.3.10 GetValidity()

```
bool TempSensor::GetValidity ( )  [inline]
```

Data validity get function.

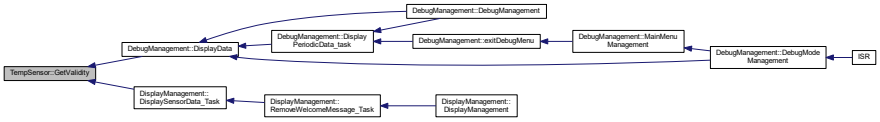
This function returns the validity of the sensor data

Returns

True if the sensor values are valid, false otherwise

Definition at line 135 of file TempSensor.h.

Here is the caller graph for this function:



3.21.3.11 readTempSensor_task()

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

Task for reading temperature and humidity values.

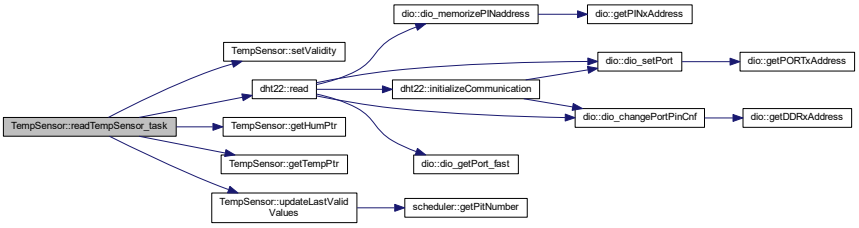
This task reads temperature and humidity data using DHT22 driver. It is called every 5 seconds.

Returns

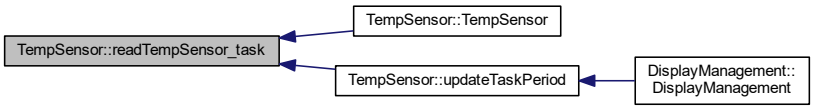
Nothing

Definition at line 44 of file TempSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.21.3.12 setValidity()

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

Set data val_validity.

This function sets the class member val_validity

Parameters

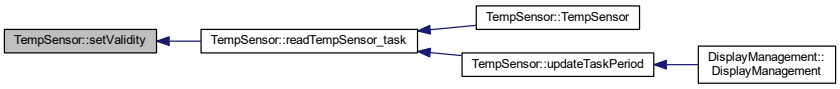
in	*validity*	Value of validity

Returns

Nothing

Definition at line 50 of file TempSensor.cpp.

Here is the caller graph for this function:

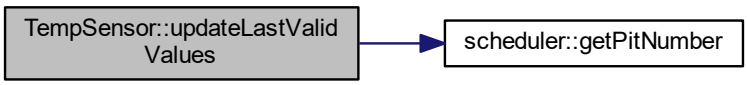


3.21.3.13 updateLastValidValues()

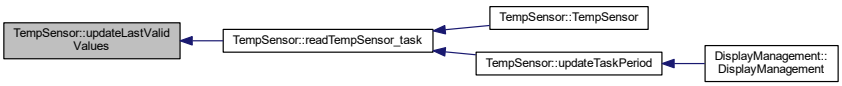
```
void TempSensor::updateLastValidValues ( )
```

Definition at line 65 of file TempSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.21.3.14 updateTaskPeriod()

```
bool TempSensor::updateTaskPeriod (
            uint16_t period )
```

Task period update.

This function updates the period of the temperature task.

Parameters

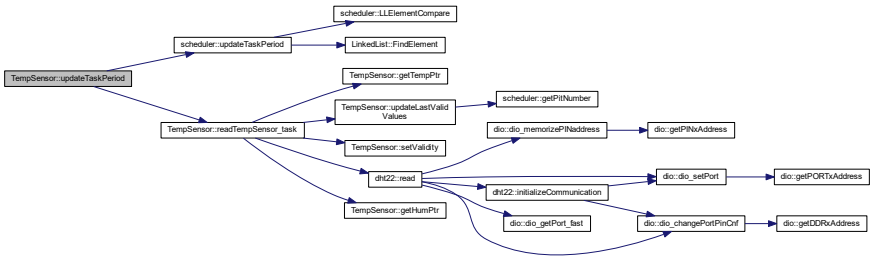
in	*period*	New period of the task

Returns

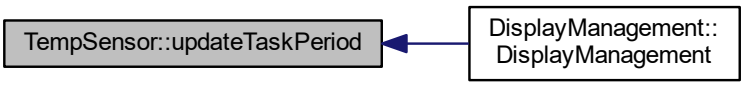
True if the period has been updated, false otherwise

Definition at line 92 of file TempSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.21.4 Member Data Documentation

3.21.4.1 read_humidity

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

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

Definition at line 163 of file TempSensor.h.

3.21.4.2 read_temperature

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

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

Definition at line 162 of file TempSensor.h.

3.21.4.3 task_period

```
uint16_t TempSensor::task_period  [private]
```

Task period

Definition at line 172 of file TempSensor.h.

3.21.4.4 valid_hum

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

Valid value of humidity

Definition at line 170 of file TempSensor.h.

3.21.4.5 valid_pit

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

pit number of the last time when data were valid

Definition at line 167 of file TempSensor.h.

3.21.4.6 valid_temp

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

Valid value of temperature

Definition at line 169 of file TempSensor.h.

3.21.4.7 validity

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

validity of official temperature and humidity data

Definition at line 166 of file TempSensor.h.

3.21.4.8 validity_last_read

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

Validity of last read temperature and humidity data

Definition at line 164 of file TempSensor.h.

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

- TempSensor.h
- TempSensor.cpp

3.22 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

3.22.1 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 19 of file timer.h.

3.22.2 Constructor & Destructor Documentation

3.22.2.1 timer()

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

Class constructor.

This function initializes class attributes

Returns

Nothing

Definition at line 15 of file timer.cpp.

3.22.3 Member Function Documentation

3.22.3.1 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 20 of file timer.cpp.

Here is the caller graph for this function:

3.22.3.2 getTimer1Value()

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

Reads current value of timer #1.

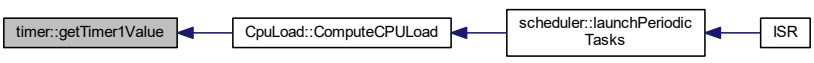
This function reads the value of of timer #1 using register TCNT1. The function is inlined to speed up SW execution.

Returns

Current timer value

Definition at line 58 of file timer.h.

Here is the caller graph for this function:



3.22.3.3 startTimer1()

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

Start Timer #1.

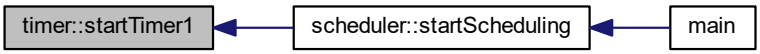
This functions starts Timer #1. Timer shall be initialized before this function is called.

Returns

Nothing

Definition at line 58 of file timer.cpp.

Here is the caller graph for this function:



3.22.3.4 stopTimer1()

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

Stops Timer #1.

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

Returns

Nothing

Definition at line 69 of file timer.cpp.

3.22.4 Member Data Documentation

3.22.4.1 prescaler

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

Definition at line 64 of file timer.h.

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

- timer.h
- timer.cpp

3.23 usart Class Reference

USART serial bus class.

`#include <usart.h>`

Public Member Functions

- usart (uint16_t a_BaudRate)
  Class usart constructor.
- void usart_sendString (String *str)
  Send a string on USART link.
- void usart_sendByte (uint8_t data)
  Send a single byte 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.

Private Member Functions

- void usart_transmit (uint8_t Data)
  USART Transmit data.

Private Attributes

- uint16_t BaudRate

3.23.1 Detailed Description

USART serial bus class.

This class defines all useful functions for USART serial bus

Definition at line 16 of file usart.h.

3.23.2 Constructor & Destructor Documentation

3.23.2.1 usart()

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

Class usart constructor.

Initializes the class and call hardware initialization function

Parameters

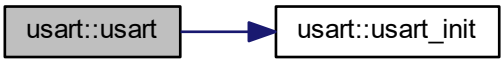
in	*a_BaudRate*	Desired Baud Rate (16 bit) - up to 57600

Returns

Nothing.

Definition at line 18 of file usart.cpp.

Here is the call graph for this function:



3.23.3 Member Function Documentation

3.23.3.1 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 74 of file usart.cpp.

3.23.3.2 usart_init()

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

USART hardware initialization.

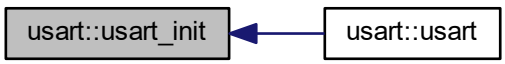
This function will initialize the USART using selected baudrate. User must pay attention to select one of the usually used Baud Rate (9600, 19200, 38400, 57600). Note that since an uint16 is used as argument, Baud rate cannot be more than 57600.

Returns

Nothing.

Definition at line 25 of file usart.cpp.

Here is the caller graph for this function:



3.23.3.3 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 90 of file usart.cpp.

3.23.3.4 usart_sendByte()

```
void usart::usart_sendByte (
            uint8_t data )
```

Send a single byte on USART link.

This function writes the given byte to the serial link using usart_transmit function

Parameters

in	*data*	Data byte being sent

Returns

Nothing.

Definition at line 68 of file usart.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

3.23.3.5 usart_sendString()

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

Send a string on USART link.

This function writes the string object data to the serial link using usart_transmit function

Parameters

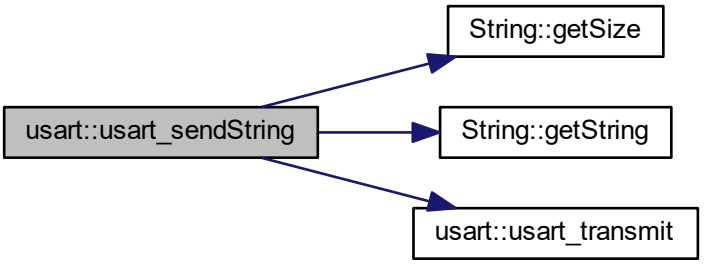
in	*str*	Pointer to the string being sent

Returns

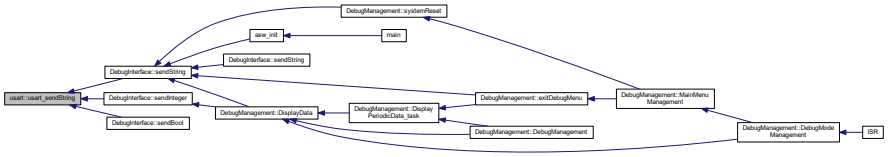
Nothing.

Definition at line 48 of file usart.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.23.3.6 usart_transmit()

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

USART Transmit data.

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

Parameters

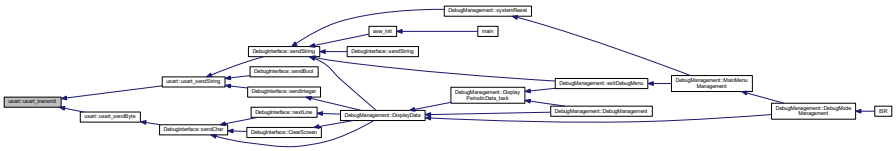
in	*Data*	Desired data char to transmit

Returns

Nothing.

Definition at line 81 of file usart.cpp.

Here is the caller graph for this function:



3.23.4 Member Data Documentation

3.23.4.1 BaudRate

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

Defines the baud rate used by driver

Definition at line 77 of file usart.h.

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

- usart.h
- usart.cpp

3.24 Watchdog Class Reference

Watchdog management class.

```
#include <Watchdog.h>
```

Public Member Functions

- Watchdog ()
  Class constructor.
- Watchdog (uint8_t timeout)
  Overloaded class constructor.
- void reset ()
  Watchdog reset function.
- void timeoutUpdate (uint8_t value)
  Watchdog timeout value update function.
- void SystemReset ()
  System reset function.
- uint16_t getTMOValue ()
  Watchdog timeout get value.

Private Member Functions

- void enable (uint8_t value)

  Watchdog enabling function.
- void disable ()

  Watchdog disabling function.

Private Attributes

- uint8_t tmo_value

3.24.1 Detailed Description

Watchdog management class.

This class provides services to manage the watchdog HW module. The watchdog shall be reset periodically to avoid a hardware reset of the system.

Definition at line 31 of file Watchdog.h.

3.24.2 Constructor & Destructor Documentation

3.24.2.1 Watchdog() [1/2]

```
Watchdog::Watchdog ( )
```

Class constructor.

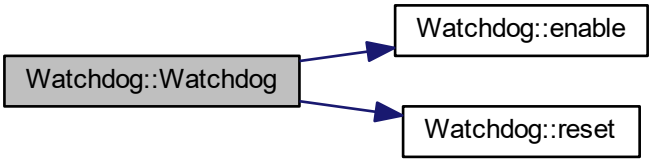
This function initializes the watchdog class. It enables the HW watchdog with a default timeout value.

Returns

Nothing

Definition at line 23 of file Watchdog.cpp.

Here is the call graph for this function:



3.24.2.2 Watchdog() [2/2]

```
Watchdog::Watchdog (
            uint8_t timeout )
```

Overloaded class constructor.

This function initializes the watchdog class. It enables the HW watchdog with the given timeout value.

Parameters

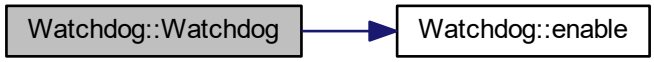
in	*timeout*	Timeout value requested for the watchdof

Returns

Nothing

Definition at line 29 of file Watchdog.cpp.

Here is the call graph for this function:



3.24.3 Member Function Documentation

3.24.3.1 disable()

```
void Watchdog::disable ( )  [private]
```

Watchdog disabling function.

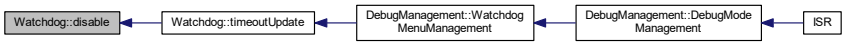
This function disables the watchdog by calling wdt_disable macro.

Returns

Nothing

Definition at line 41 of file Watchdog.cpp.

Here is the caller graph for this function:



3.24.3.2 enable()

```
void Watchdog::enable (
            uint8_t value )  [private]
```

Watchdog enabling function.

This function enables the watchdog by calling wdt_enable macro.

Parameters

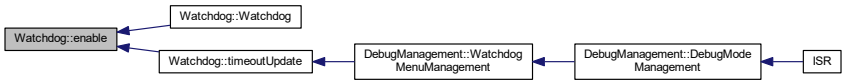
in	*value*	Timeout value

Returns

Nothing

Definition at line 35 of file Watchdog.cpp.

Here is the caller graph for this function:



3.24.3.3 getTMOValue()

```
uint16_t Watchdog::getTMOValue ( )
```

Watchdog timeout get value.

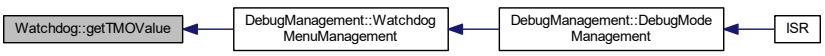
This function returns the current watchdog timeout value in ms. It has to convert the value of tmo_value into a numeric value of the timeout.

Returns

Timeout value.

Definition at line 63 of file Watchdog.cpp.

Here is the caller graph for this function:



3.24.3.4 reset()

```
void Watchdog::reset ( )
```

Watchdog reset function.

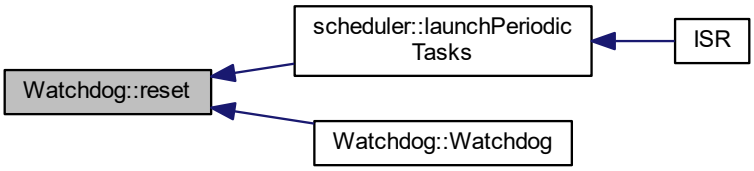
This function resets the watchdog timer by calling wdt_reset macro

Returns

Nothing

Definition at line 46 of file Watchdog.cpp.

Here is the caller graph for this function:



3.24.3.5 SystemReset()

```
void Watchdog::SystemReset ( )
```

System reset function.

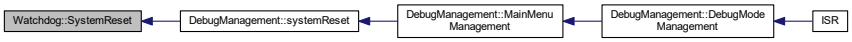
This function provokes a system reset by going in an infinite loop. Thus the watchdog will reset the CPU when the timeout occurs.

Returns

Nothing

Definition at line 58 of file Watchdog.cpp.

Here is the caller graph for this function:



3.24.3.6 timeoutUpdate()

```
void Watchdog::timeoutUpdate (
            uint8_t value )
```

Watchdog timeout value update function.

This function updates the timeout value of the watchdog. It disables then re-enables the watchdog.

Parameters

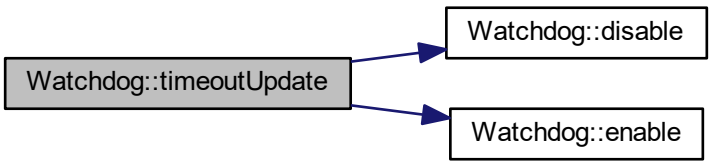
in	*value*	New timeout value

Returns

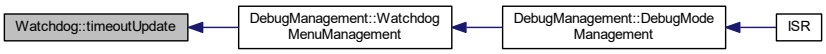
Nothing

Definition at line 51 of file Watchdog.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.24.4 Member Data Documentation

3.24.4.1 tmo_value

```
uint8_t Watchdog::tmo_value  [private]
```

Current timeout value

Definition at line 87 of file Watchdog.h.

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

- Watchdog.h
- Watchdog.cpp

Chapter 4

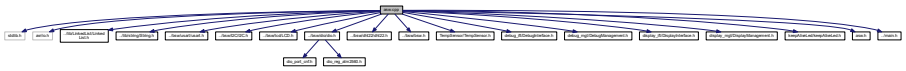
File Documentation

4.1 asw.cpp File Reference

ASW main file.

```
#include <stdlib.h>
#include <avr/io.h>
#include "../lib/LinkedList/LinkedList.h"
#include "../lib/string/String.h"
#include "../bsw/usart/usart.h"
#include "../bsw/I2C/I2C.h"
#include "../bsw/lcd/LCD.h"
#include "../bsw/dio/dio.h"
#include "../bsw/dht22/dht22.h"
#include "../bsw/bsw.h"
#include "TempSensor/TempSensor.h"
#include "debug_ift/DebugInterface.h"
#include "debug_mgt/DebugManagement.h"
#include "display_ift/DisplayInterface.h"
#include "display_mgt/DisplayManagement.h"
#include "keepAliveLed/keepAliveLed.h"
#include "asw.h"
#include "../main.h"
```

Include dependency graph for asw.cpp:



Functions

- void asw_init ()

  Initialization of ASW.

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.

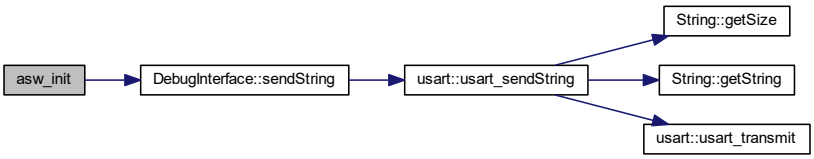
This function instantiates all applicative objects. Some objects are not created by this function but directly by the upper-level class. The addresses of objects are then stored in ASW_cnf_struct structure.
The debug interface obkect is created only if the debug pin is set to logical high level.
This function shall be called after BSW initialization function.

Returns

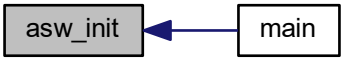
Nothing

Definition at line 37 of file asw.cpp.

Here is the call graph for this function:



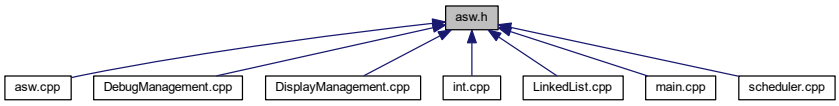
Here is the caller graph for this function:



4.2 asw.h File Reference

ASW main header file.

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



Classes

- struct T_ASW_init_cnf

  ASW initialization configuration structure.

Functions

- void asw_init ()

  Initialization of ASW.

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.

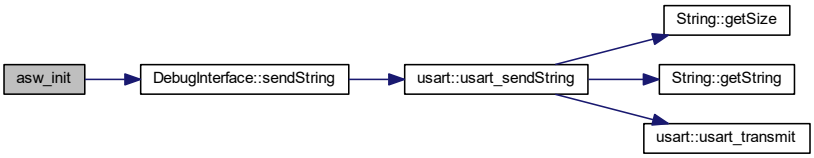
This function instantiates all applicative objects. Some objects are not created by this function but directly by the upper-level class. The addresses of objects are then stored in ASW_cnf_struct structure.
The debug interface obkect is created only if the debug pin is set to logical high level.
This function shall be called after BSW initialization function.

Returns

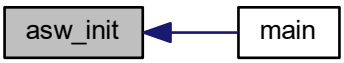
Nothing

Definition at line 37 of file asw.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



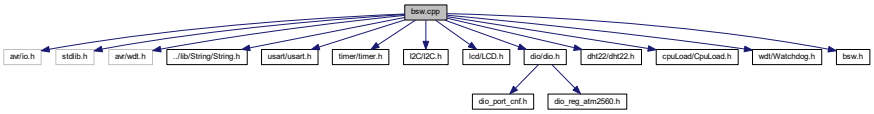
4.3 bsw.cpp File Reference

BSW main file.

```
#include <avr/io.h>
#include <stdlib.h>
#include <avr/wdt.h>
#include "../lib/String/String.h"
#include "usart/usart.h"
#include "timer/timer.h"
#include "I2C/I2C.h"
#include "lcd/LCD.h"
```

```
#include "dio/dio.h"
#include "dht22/dht22.h"
#include "cpuLoad/CpuLoad.h"
#include "wdt/Watchdog.h"
#include "bsw.h"
```

Include dependency graph for bsw.cpp:



Functions

- void bsw_init ()

  Initialization of BSW.

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.

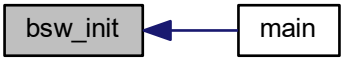
This function instantiates all driver objects, leading hardware initialization. The addresses of driver objects are then stored in BSW_cnf_struct structure.

Returns

Nothing

Definition at line 26 of file bsw.cpp.

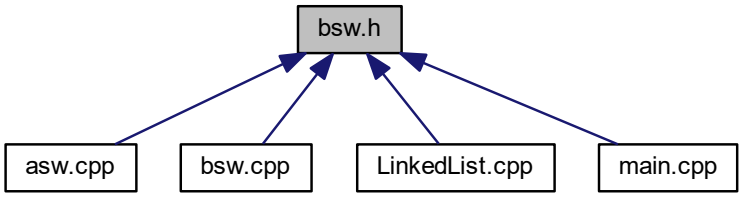
Here is the caller graph for this function:



4.4 bsw.h File Reference

BSW main header file.

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



Functions

- void bsw_init ()

  Initialization of BSW.

4.4.1 Detailed Description

BSW main header file.

Date

13 mars 2018

Author

nicls67

4.4.2 Function Documentation

4.4.2.1 bsw_init()

```
void bsw_init ( )
```

Initialization of BSW.

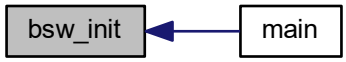
This function instantiates all driver objects, leading hardware initialization. The addresses of driver objects are then stored in BSW_cnf_struct structure.

Returns

Nothing

Definition at line 26 of file bsw.cpp.

Here is the caller graph for this function:

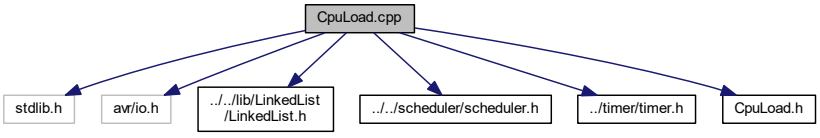


4.5 CpuLoad.cpp File Reference

Defines functions of class CpuLoad.

```
#include <stdlib.h>
#include <avr/io.h>
#include "../../lib/LinkedList/LinkedList.h"
#include "../../scheduler/scheduler.h"
#include "../timer/timer.h"
#include "CpuLoad.h"
```

Include dependency graph for CpuLoad.cpp:



Variables

- CpuLoad * p_global_BSW_cpuload

4.5.1 Detailed Description

Defines functions of class CpuLoad.

Date

21 mars 2019

Author

nicls67

4.5.2 Variable Documentation

4.5.2.1 p_global_BSW_cpuload

```
CpuLoad* p_global_BSW_cpuload
```

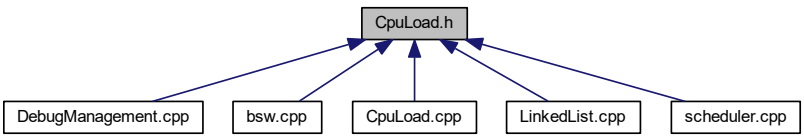
Pointer to cpu load library object

Definition at line 18 of file CpuLoad.cpp.

4.6 CpuLoad.h File Reference

CpuLoad class header file.

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

Variables

- CpuLoad * p_global_BSW_cpuload

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.

4.6.3 Variable Documentation

4.6.3.1 p_global_BSW_cpuload

```
CpuLoad* p_global_BSW_cpuload
```

Pointer to cpu load library object

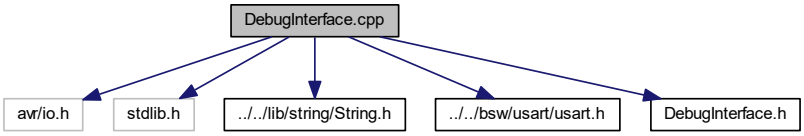
Definition at line 18 of file CpuLoad.cpp.

4.7 DebugInterface.cpp File Reference

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

```
#include <avr/io.h>
#include <stdlib.h>
#include "../../lib/string/String.h"
#include "../../bsw/usart/usart.h"
#include "DebugInterface.h"
```

Include dependency graph for DebugInterface.cpp:



Variables

- DebugInterface * p_global_ASW_DebugInterface

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 p_global_ASW_DebugInterface

```
DebugInterface* p_global_ASW_DebugInterface
```

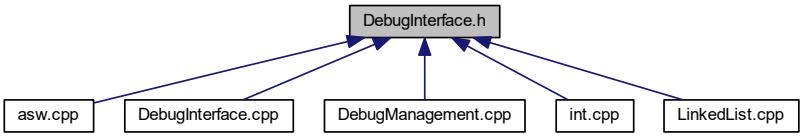
Pointer to USART debug interface object

Definition at line 19 of file DebugInterface.cpp.

4.8 DebugInterface.h File Reference

Header file for debug and logging functions.

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



Classes

- class DebugInterface

  Class used for debugging on usart link.

Macros

- #define USART_BAUDRATE (uint16_t)9600

Variables

- DebugInterface * p_global_ASW_DebugInterface

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 USART_BAUDRATE

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

usart connection to PC uses a baud rate of 9600

Definition at line 15 of file DebugInterface.h.

4.8.3 Variable Documentation

4.8.3.1 p_global_ASW_DebugInterface

```
DebugInterface* p_global_ASW_DebugInterface
```

Pointer to USART debug interface object

Definition at line 19 of file DebugInterface.cpp.

4.9 DebugManagement.cpp File Reference

Debug management class source file.

```
#include <avr/io.h>
#include <stdlib.h>
#include <avr/wdt.h>
#include "../../lib/string/String.h"
#include "../../lib/LinkedList/LinkedList.h"
#include "../../scheduler/scheduler.h"
#include "../../bsw/usart/usart.h"
#include "../../bsw/cpuLoad/CpuLoad.h"
#include "../../bsw/wdt/Watchdog.h"
#include "../TempSensor/TempSensor.h"
#include "../debug_ift/DebugInterface.h"
#include "DebugManagement.h"
#include "../asw.h"
#include "../../main.h"
```

Include dependency graph for DebugManagement.cpp:

Variables

- DebugManagement * p_global_ASW_DebugManagement
- const uint8_t str_debug_main_menu [ ]

  Main menu of debug mode.
- const uint8_t str_debug_wdg_menu [ ]

  Watchdog menu of debug mode.
- const uint8_t str_debug_wdg_timeout_update_selection [ ]

  Watchdog timeout update selection.
- const uint8_t str_debug_info_message_wrong_menu_selection [ ] = "Impossible de faire ca... !"

  Info menu string in case a wrong selection has been performed.
- const uint8_t str_debug_info_message_wdg_tmo_updated [ ] = "Valeur modifiee !"

  Info menu string in case the watchdog timeout value has been updated.
- const uint8_t str_debug_info_message_wdg_tmo_value [ ] = "Valeur du timeout watchdog (ms) : "

  Info menu string displaying the current value of the watchdog timeout.

4.9.1 Detailed Description

Debug management class source file.

Date

8 mai 2019

Author

nicls67

4.9.2 Variable Documentation

4.9.2.1 p_global_ASW_DebugManagement

```
DebugManagement* p_global_ASW_DebugManagement
```

Pointer to the DebugManagement object

Definition at line 30 of file DebugManagement.cpp.

4.9.2.2 str_debug_info_message_wdg_tmo_updated

```
const uint8_t str_debug_info_message_wdg_tmo_updated[] = "Valeur modifiee !"
```

Info menu string in case the watchdog timeout value has been updated.

Definition at line 78 of file DebugManagement.cpp.

4.9.2.3 str_debug_info_message_wdg_tmo_value

```
const uint8_t str_debug_info_message_wdg_tmo_value[] = "Valeur du timeout watchdog (ms) :  "
```

Info menu string displaying the current value of the watchdog timeout.

Definition at line 83 of file DebugManagement.cpp.

4.9.2.4 str_debug_info_message_wrong_menu_selection

```
const uint8_t str_debug_info_message_wrong_menu_selection[] = "Impossible de faire ca...  !"
```

Info menu string in case a wrong selection has been performed.

Definition at line 73 of file DebugManagement.cpp.

4.9.2.5 str_debug_main_menu

```
const uint8_t str_debug_main_menu[]
```

Initial value:

```
=
        "Menu principal :  \n"
        "    1 : Watchdog\n"
        "\n"
        "    r : Reset du systeme\n"
        "    q : Quitter debug\n"
```

Main menu of debug mode.

Definition at line 35 of file DebugManagement.cpp.

4.9.2.6 str_debug_wdg_menu

```
const uint8_t str_debug_wdg_menu[]
```

Initial value:

```
=
        "Menu watchdog : \n"
        "    1 : Changer timeout\n"
        "    2 : Afficher valeur actuelle du timeout\n"
        "\n"
        "    q : Retour\n"
```

Watchdog menu of debug mode.

Definition at line 45 of file DebugManagement.cpp.

4.9.2.7 str_debug_wdg_timeout_update_selection

```
const uint8_t str_debug_wdg_timeout_update_selection[]
```

Initial value:

```
=
        "Selection du timeout watchdog : \n"
        "    0 : 15 ms\n"
        "    1 : 30 ms\n"
        "    2 : 60 ms\n"
        "    3 : 120 ms\n"
        "    4 : 250 ms\n"
        "    5 : 500 ms\n"
        "    6 : 1 s\n"
        "    7 : 2 s\n"
        "    8 : 4 s\n"
        "    9 : 8 s\n"
        "\n"
        "    a : Annuler\n"
```

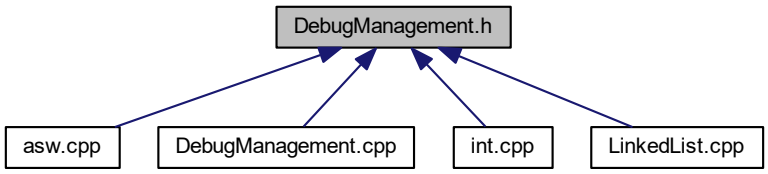
Watchdog timeout update selection.

Definition at line 55 of file DebugManagement.cpp.

4.10 DebugManagement.h File Reference

Debug management class header file.

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



Classes

- struct debug_mgt_state_struct_t
  Structure containing all debug states.
- class DebugManagement
  Debug management class.

Macros

- #define PERIOD_MS_TASK_DISPLAY_DEBUG_DATA 5000
- #define PERIOD_MS_TASK_DISPLAY_CPU_LOAD 5000

Enumerations

- enum debug_mgt_main_menu_state_t { MAIN_MENU, WDG_MENU }
  Defines the debug states.
- enum debug_mgt_wdg_state_t { WDG_MAIN, WDG_TMO_UPDATE }
  Defines possible states for watchdog management.

Variables

- DebugManagement * p_global_ASW_DebugManagement

4.10.1 Detailed Description

Debug management class header file.

Date

8 mai 2019

Author

nicls67

4.10.2 Macro Definition Documentation

4.10.2.1 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 DebugManagement.h.

4.10.2.2 PERIOD_MS_TASK_DISPLAY_DEBUG_DATA

```
#define PERIOD_MS_TASK_DISPLAY_DEBUG_DATA 5000
```

Period for displaying temperature and humidity data

Definition at line 13 of file DebugManagement.h.

4.10.3 Enumeration Type Documentation

4.10.3.1 debug_mgt_main_menu_state_t

```
enum debug_mgt_main_menu_state_t
```

Defines the debug states.

Enumerator

MAIN_MENU	Init state : main menu is displayed
WDG_MENU	Watchdog state : watchdog menu is displayed

Definition at line 20 of file DebugManagement.h.

4.10.3.2 debug_mgt_wdg_state_t

```
enum debug_mgt_wdg_state_t
```

Defines possible states for watchdog management.

Enumerator

WDG_MAIN	Main menu of watchdog management
WDG_TMO_UPDATE	Timeout update mode

Definition at line 30 of file DebugManagement.h.

4.10.4 Variable Documentation

4.10.4.1 p_global_ASW_DebugManagement

```
DebugManagement* p_global_ASW_DebugManagement
```

Pointer to the DebugManagement object

Definition at line 30 of file DebugManagement.cpp.

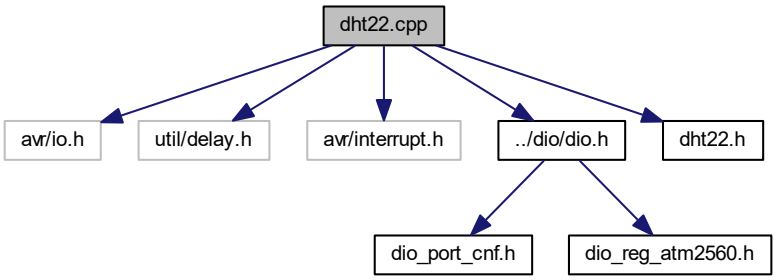
4.11 dht22.cpp File Reference

This file defines classes for DHT22 driver.

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

```
#include "dht22.h"
```

Include dependency graph for dht22.cpp:



Macros

- #define MAX_WAIT_TIME_US 100

Variables

- dht22 * p_global_BSW_dht22

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
```

Maximum waiting time in microseconds

Definition at line 18 of file dht22.cpp.

4.11.3 Variable Documentation

4.11.3.1 p_global_BSW_dht22

```
dht22* p_global_BSW_dht22
```

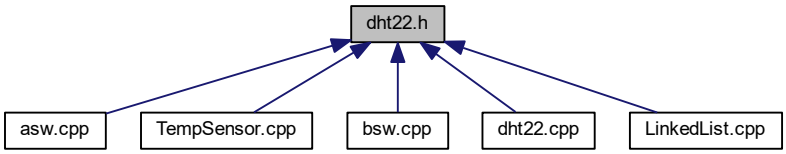
Pointer to dht22 driver object

Definition at line 20 of file dht22.cpp.

4.12 dht22.h File Reference

DHT22 driver header file.

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



Classes

- class dht22
  DHT 22 driver class.

Variables

- dht22 * p_global_BSW_dht22

4.12.1 Detailed Description

DHT22 driver header file.

Date

23 mars 2018

Author

nicls67

4.12.2 Variable Documentation

4.12.2.1 p_global_BSW_dht22

```
dht22* p_global_BSW_dht22
```

Pointer to dht22 driver object

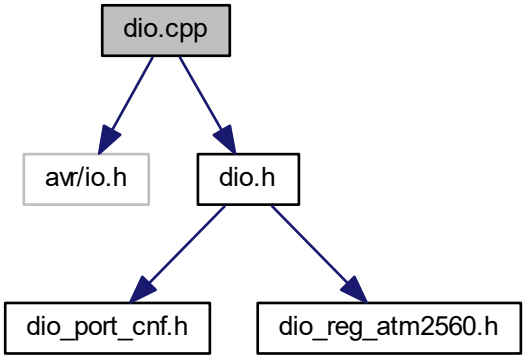
Definition at line 20 of file dht22.cpp.

4.13 dio.cpp File Reference

DIO library.

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

Include dependency graph for dio.cpp:



Variables

- dio * p_global_BSW_dio

4.13.1 Detailed Description

DIO library.

Date

13 mars 2018

Author

nicls67

4.13.2 Variable Documentation

4.13.2.1 p_global_BSW_dio

```
dio* p_global_BSW_dio
```

Pointer to dio driver object

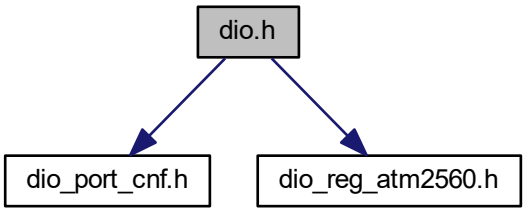
Definition at line 14 of file dio.cpp.

4.14 dio.h File Reference

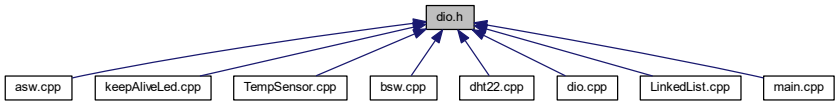
DIO library header file.

```
#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)

Variables

- dio * p_global_BSW_dio

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 20 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 19 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 18 of file dio.h.

4.14.2.4 PORT_CNF_IN

```
#define PORT_CNF_IN 0
```

Pin is configured as input

Definition at line 16 of file dio.h.

4.14.2.5 PORT_CNF_OUT

```
#define PORT_CNF_OUT 1
```

Pin is configured as output

Definition at line 15 of file dio.h.

4.14.3 Variable Documentation

4.14.3.1 p_global_BSW_dio

```
dio* p_global_BSW_dio
```

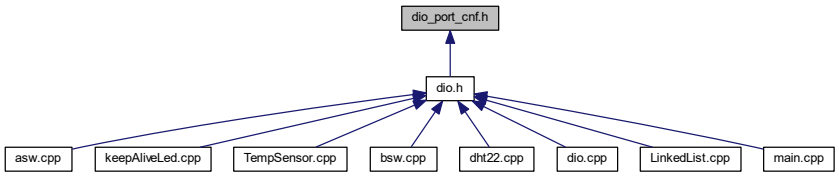
Pointer to dio driver object

Definition at line 14 of file dio.cpp.

4.15 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)0b01010000

  Defines the configuration of PORTB register.
- #define PORT_A 0
- #define PORT_B 1
- #define PORT_C 2
- #define PORT_D 3

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 PORT_A

```
#define PORT_A 0
```

PORTA index

Definition at line 42 of file dio_port_cnf.h.

4.15.2.2 PORT_B

```
#define PORT_B 1
```

PORTB index

Definition at line 43 of file dio_port_cnf.h.

4.15.2.3 PORT_C

```
#define PORT_C 2
```

PORTC index

Definition at line 44 of file dio_port_cnf.h.

4.15.2.4 PORT_D

```
#define PORT_D 3
```

PORTD index

Definition at line 45 of file dio_port_cnf.h.

4.15.2.5 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 : IN
PB5 : N/A
PB6 : OUT
PB7 : OUT

Definition at line 25 of file dio_port_cnf.h.

4.15.2.6 PORTB_CNF_PORTB

```
#define PORTB_CNF_PORTB (uint8_t)0b01010000
```

Defines the configuration of PORTB register.

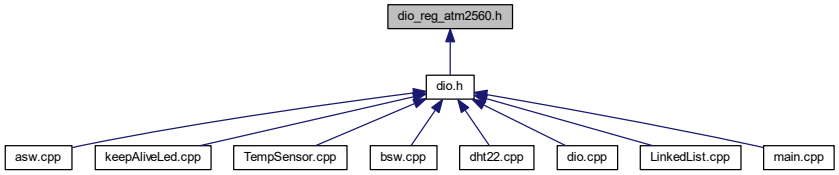
This constant defines the initial state of IO pins for PORT B. It will configure register PORTB. For outputs pins, it defines the initial level (high or low). For input pins, it defines if the pins is configured as high-Z or pull-up.
PB0 : N/A
PB1 : N/A
PB2 : N/A
PB3 : N/A
PB4 : Pull-up
PB5 : N/A
PB6 : HIGH
PB7 : LOW

Definition at line 40 of file dio_port_cnf.h.

4.16 dio_reg_atm2560.h File Reference

Defines DIO register addresses for ATMEGA2560.

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 Detailed Description

Defines DIO register addresses for ATMEGA2560.

Date

19 mars 2019

Author

nicls67

4.16.2 Macro Definition Documentation

4.16.2.1 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.

4.16.2.2 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.

4.16.2.3 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.

4.16.2.4 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.

4.16.2.5 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.

4.16.2.6 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.

4.16.2.7 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.

4.16.2.8 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.

4.16.2.9 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.

4.16.2.10 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.

4.16.2.11 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.

4.16.2.12 PORTD_PTR

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

Macro defining pointer to PORT D register

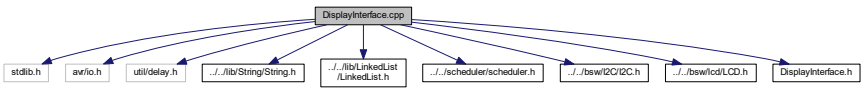
Definition at line 17 of file dio_reg_atm2560.h.

4.17 DisplayInterface.cpp File Reference

Source code file for display services.

```
#include <stdlib.h>
#include <avr/io.h>
#include <util/delay.h>
#include "../../lib/String/String.h"
#include "../../lib/LinkedList/LinkedList.h"
#include "../../scheduler/scheduler.h"
#include "../../bsw/I2C/I2C.h"
#include "../../bsw/lcd/LCD.h"
#include "DisplayInterface.h"
```

Include dependency graph for DisplayInterface.cpp:



Variables

- DisplayInterface * p_global_ASW_DisplayInterface

4.17.1 Detailed Description

Source code file for display services.

Date

23 avr. 2019

Author

nicls67

4.17.2 Variable Documentation

4.17.2.1 p_global_ASW_DisplayInterface

```
DisplayInterface* p_global_ASW_DisplayInterface
```

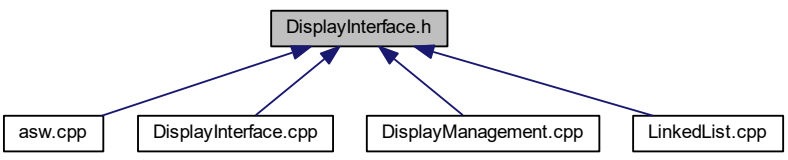
Pointer to DisplayInterface object

Definition at line 25 of file DisplayInterface.cpp.

4.18 DisplayInterface.h File Reference

DisplayInterface class header file.

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



Classes

- struct T_Display_shift_data
  Structure containing shift data.
- struct T_display_data
  Structure containing display data.
- class DisplayInterface
  Display interface services class.

Macros

- #define DISPLAY_LINE_SHIFT_PERIOD_MS 500
- #define DISPLAY_LINE_SHIFT_TEMPO_TIME 6

Enumerations

- enum T_DisplayInterface_LineDisplayMode { NORMAL, LINE_SHIFT, GO_TO_NEXT_LINE }
  Modes for line display.
- enum T_DisplayInterface_LineAlignment { LEFT, CENTER, RIGHT }
  Alignment mode for line display.

Variables

- DisplayInterface * p_global_ASW_DisplayInterface

4.18.1 Detailed Description

DisplayInterface class header file.

Date

23 avr. 2019

Author

nicls67

4.18.2 Macro Definition Documentation

4.18.2.1 DISPLAY_LINE_SHIFT_PERIOD_MS

```
#define DISPLAY_LINE_SHIFT_PERIOD_MS 500
```

In "line shift" mode for line display, line is shifted every 500 ms

Definition at line 68 of file DisplayInterface.h.

4.18.2.2 DISPLAY_LINE_SHIFT_TEMPO_TIME

```
#define DISPLAY_LINE_SHIFT_TEMPO_TIME 6
```

In "line shift" mode for line display, a temporization of 6 periods is added at the end and the beginning of the lines

Definition at line 69 of file DisplayInterface.h.

4.18.3 Enumeration Type Documentation

4.18.3.1 T_DisplayInterface_LineAlignment

```
enum T_DisplayInterface_LineAlignment
```

Alignment mode for line display.

This enumeration defines the possible alignment mode for the text displayed. It is only used when the display mode is NORMAL or GO_TO_NEXT_LINE.

Enumerator

LEFT	Text is aligned left
CENTER	Text is centered
RIGHT	Text is aligned right

Definition at line 33 of file DisplayInterface.h.

4.18.3.2 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 20 of file DisplayInterface.h.

4.18.4 Variable Documentation

4.18.4.1 p_global_ASW_DisplayInterface

```
DisplayInterface* p_global_ASW_DisplayInterface
```

Pointer to DisplayInterface object

Definition at line 25 of file DisplayInterface.cpp.

4.19 DisplayManagement.cpp File Reference

Display management source file.

```
#include <stdlib.h>
#include <avr/io.h>
#include "../../lib/LinkedList/LinkedList.h"
#include "../../lib/string/string.h"
#include "../../scheduler/scheduler.h"
#include "../../bsw/I2C/I2C.h"
#include "../../bsw/lcd/LCD.h"
#include "../TempSensor/TempSensor.h"
#include "../display_ift/DisplayInterface.h"
#include "DisplayManagement.h"
#include "../asw.h"
#include "../../main.h"
```

Include dependency graph for DisplayManagement.cpp:

Variables

- DisplayManagement * p_global_ASW_DisplayManagement

4.19.1 Detailed Description

Display management source file.

Date

1 mai 2019

Author

nicls67

4.19.2 Variable Documentation

4.19.2.1 p_global_ASW_DisplayManagement

```
DisplayManagement* p_global_ASW_DisplayManagement
```

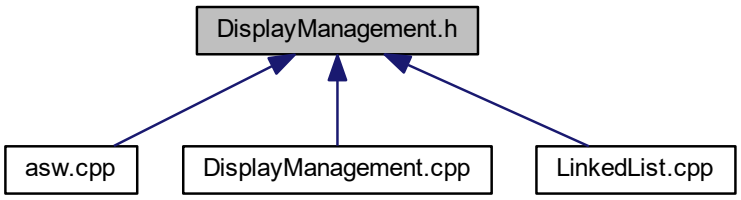
Pointer to DisplayManagement object

Definition at line 28 of file DisplayManagement.cpp.

4.20 DisplayManagement.h File Reference

Display management class header file.

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



Classes

- class DisplayManagement

  Display management class.

Macros

- #define DISPLAY_MGT_LCD_I2C_ADDR 0x27
- #define DISPLAY_MGT_PERIOD_TASK_SENSOR 1500
- #define DISPLAY_MGT_PERIOD_WELCOME_MSG_REMOVAL 5000
- #define DISPLAY_MGT_LINE_TEMP 0
- #define DISPLAY_MGT_LINE_HUM 1
- #define DISPLAY_MGT_I2C_BITRATE (uint32_t)100000

Variables

- const T_LCD_conf_struct LCD_init_cnf
  LCD configuration structure.
- const uint8_t welcomeMessageString [ ] = "Bienvenue !"
- const uint8_t tempDisplayString [ ] = "Temperature : "
- const uint8_t humidityDisplayString [ ] = "Humidite : "
- const uint8_t noSensorDisplayString [ ] = "Capteur de temperature desactive"
- DisplayManagement * p_global_ASW_DisplayManagement

4.20.1 Detailed Description

Display management class header file.

Date

1 mai 2019

Author

nicls67

4.20.2 Macro Definition Documentation

4.20.2.1 DISPLAY_MGT_I2C_BITRATE

```
#define DISPLAY_MGT_I2C_BITRATE (uint32_t)100000
```

I2C bus bitrate is 100 kHz

Definition at line 21 of file DisplayManagement.h.

4.20.2.2 DISPLAY_MGT_LCD_I2C_ADDR

```
#define DISPLAY_MGT_LCD_I2C_ADDR 0x27
```

I2C address of the screen

Definition at line 13 of file DisplayManagement.h.

4.20.2.3 DISPLAY_MGT_LINE_HUM

```
#define DISPLAY_MGT_LINE_HUM 1
```

Current humidity is displayed on line 1

Definition at line 19 of file DisplayManagement.h.

4.20.2.4 DISPLAY_MGT_LINE_TEMP

```
#define DISPLAY_MGT_LINE_TEMP 0
```

Current temperature is displayed on line 0

Definition at line 18 of file DisplayManagement.h.

4.20.2.5 DISPLAY_MGT_PERIOD_TASK_SENSOR

```
#define DISPLAY_MGT_PERIOD_TASK_SENSOR 1500
```

Display is updated every 1.5s

Definition at line 15 of file DisplayManagement.h.

4.20.2.6 DISPLAY_MGT_PERIOD_WELCOME_MSG_REMOVAL

```
#define DISPLAY_MGT_PERIOD_WELCOME_MSG_REMOVAL 5000
```

Time after which one the welcome message is removed

Definition at line 16 of file DisplayManagement.h.

4.20.3 Variable Documentation

4.20.3.1 humidityDisplayString

```
const uint8_t humidityDisplayString[] = "Humidite :  "
```

String used for humidity display

Definition at line 43 of file DisplayManagement.h.

4.20.3.2 LCD_init_cnf

```
const T_LCD_conf_struct LCD_init_cnf
```

Initial value:

```
= {
    DISPLAY_MGT_I2C_BITRATE,
    DISPLAY_MGT_LCD_I2C_ADDR,
    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
}
```

LCD configuration structure.

This structure defines the initial configuration of the LCD screen.

Definition at line 27 of file DisplayManagement.h.

4.20.3.3 noSensorDisplayString

```
const uint8_t noSensorDisplayString[] = "Capteur de temperature desactive"
```

String used in case temperature sensor is deactivated

Definition at line 44 of file DisplayManagement.h.

4.20.3.4 p_global_ASW_DisplayManagement

```
DisplayManagement* p_global_ASW_DisplayManagement
```

Pointer to DisplayManagement object

Definition at line 28 of file DisplayManagement.cpp.

4.20.3.5 tempDisplayString

```
const uint8_t tempDisplayString[] = "Temperature :  "
```

String used for temperature display

Definition at line 42 of file DisplayManagement.h.

4.20.3.6 welcomeMessageString

```
const uint8_t welcomeMessageString[] = "Bienvenue !"
```

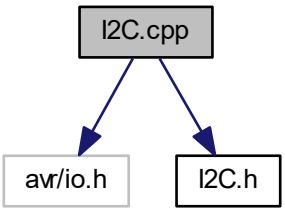
Definition at line 41 of file DisplayManagement.h.

4.21 I2C.cpp File Reference

Two-wire interface (I2C) source file.

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

Include dependency graph for I2C.cpp:



Variables

- I2C * p_global_BSW_i2c

4.21.1 Detailed Description

Two-wire interface (I2C) source file.

Date

19 avr. 2019

Author

nicls67

4.21.2 Variable Documentation

4.21.2.1 p_global_BSW_i2c

```
I2C* p_global_BSW_i2c
```

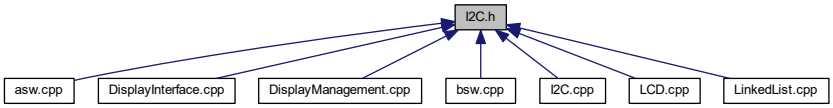
Pointer to I2C driver object

Definition at line 14 of file I2C.cpp.

4.22 I2C.h File Reference

I2C class header file.

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

Variables

- I2C * p_global_BSW_i2c

4.22.1 Detailed Description

I2C class header file.

Date

19 avr. 2019

Author

nicls67

4.22.2 Macro Definition Documentation

4.22.2.1 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.

4.22.2.2 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.

4.22.2.3 START

```
#define START 0x08
```

TWSR status code : START condition transmitted

Definition at line 13 of file I2C.h.

4.22.3 Variable Documentation

4.22.3.1 p_global_BSW_i2c

```
I2C* p_global_BSW_i2c
```

Pointer to I2C driver object

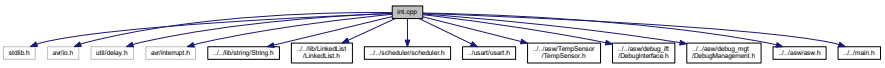
Definition at line 14 of file I2C.cpp.

4.23 int.cpp File Reference

Interrupt management source file.

```
#include <stdlib.h>
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "../../lib/string/String.h"
#include "../../lib/LinkedList/LinkedList.h"
#include "../../scheduler/scheduler.h"
#include "../usart/usart.h"
#include "../../asw/TempSensor/TempSensor.h"
#include "../../asw/debug_ift/DebugInterface.h"
#include "../../asw/debug_mgt/DebugManagement.h"
#include "../../asw/asw.h"
#include "../../main.h"
```

Include dependency graph for int.cpp:



Functions

- ISR (TIMER1_COMPA_vect)

  Main software interrupt.
- ISR (USART0_RX_vect)

  USART Rx Complete interrupt.

4.23.1 Detailed Description

Interrupt management source file.

Date

22 mai 2019

Author

nicls67

4.23.2 Function Documentation

4.23.2.1 ISR() [1/2]

```
ISR (
            TIMER1_COMPA_vect  )
```

Main software interrupt.

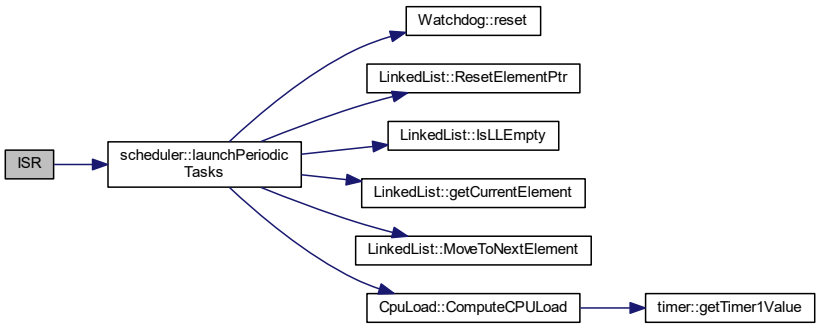
This function handles the interrupt raised by Timer #1. It wakes up the software every 500 ms to perform applications.

Returns

Nothing

Definition at line 34 of file int.cpp.

Here is the call graph for this function:



4.23.2.2 ISR() [2/2]

```
ISR (
            USART0_RX_vect  )
```

USART Rx Complete interrupt.

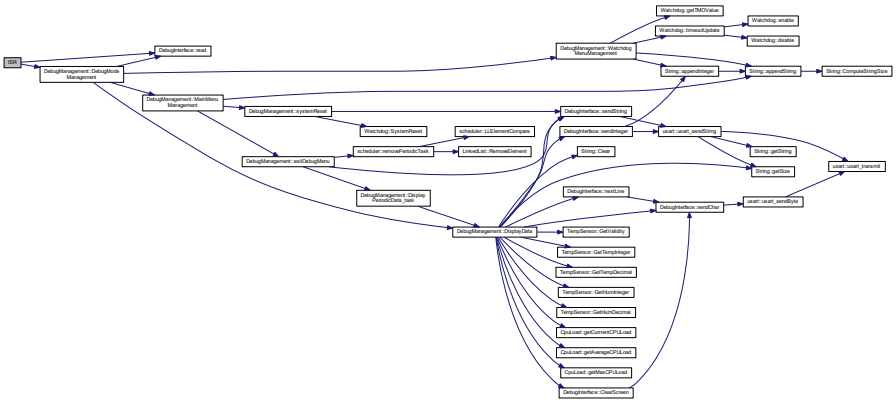
This function handles the interrupt raised when a frame has been received by USART. If debug mode mode is active, it calls debug mode management function. If inactive, it calls debug mode activation function if the received character is 'a'

Returns

Nothing

Definition at line 46 of file int.cpp.

Here is the call graph for this function:

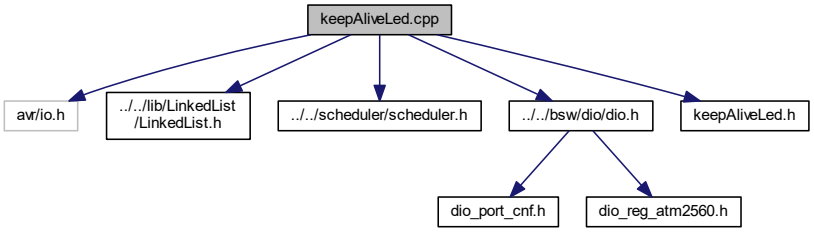


4.24 keepAliveLed.cpp File Reference

Definition of function for class keepAliveLed.

```
#include <avr/io.h>
#include "../../lib/LinkedList/LinkedList.h"
#include "../../scheduler/scheduler.h"
#include "../../bsw/dio/dio.h"
#include "keepAliveLed.h"
```

Include dependency graph for keepAliveLed.cpp:



Variables

- keepAliveLed * p_global_ASW_keepAliveLed

4.24.1 Detailed Description

Definition of function for class keepAliveLed.

Date

17 mars 2018

Author

nicls67

4.24.2 Variable Documentation

4.24.2.1 p_global_ASW_keepAliveLed

```
keepAliveLed* p_global_ASW_keepAliveLed
```

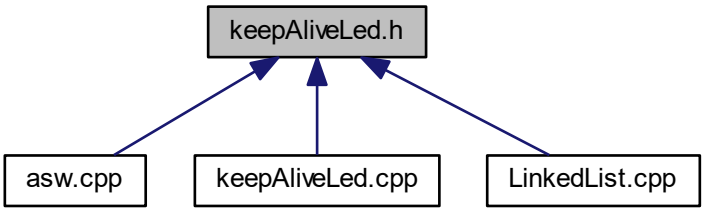
Pointer to keepAliveLed object

Definition at line 20 of file keepAliveLed.cpp.

4.25 keepAliveLed.h File Reference

Class keepAliveLed header file.

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)

Variables

- keepAliveLed * p_global_ASW_keepAliveLed

4.25.1 Detailed Description

Class keepAliveLed header file.

Date

17 mars 2018

Author

nicls67

4.25.2 Macro Definition Documentation

4.25.2.1 LED_PORT

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

LED is connected to port PB7

Definition at line 16 of file keepAliveLed.h.

4.25.2.2 PERIOD_MS_TASK_LED

```
#define PERIOD_MS_TASK_LED SW_PERIOD_MS
```

Period for led blinking

Definition at line 15 of file keepAliveLed.h.

4.25.3 Variable Documentation

4.25.3.1 p_global_ASW_keepAliveLed

```
keepAliveLed* p_global_ASW_keepAliveLed
```

Pointer to keepAliveLed object

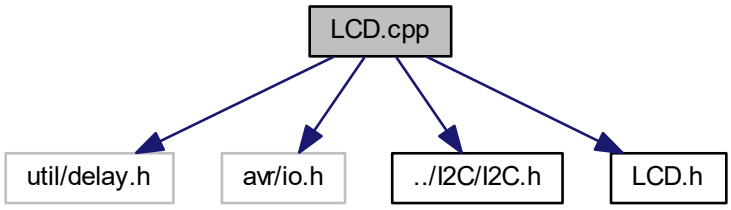
Definition at line 20 of file keepAliveLed.cpp.

4.26 LCD.cpp File Reference

LCD class source file.

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

Include dependency graph for LCD.cpp:



Variables

- LCD * p_global_BSW_lcd

4.26.1 Detailed Description

LCD class source file.

Date

20 avr. 2019

Author

nicls67

4.26.2 Variable Documentation

4.26.2.1 p_global_BSW_lcd

```
LCD* p_global_BSW_lcd
```

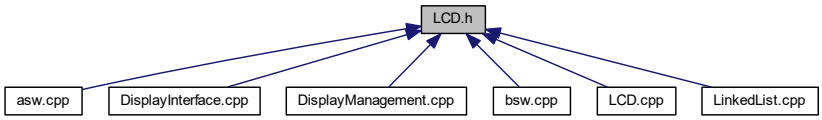
Pointer to LCD driver object

Definition at line 16 of file LCD.cpp.

4.27 LCD.h File Reference

LCD class header file.

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



Classes

- struct T_LCD_conf_struct
  Structure defining LCD configuration.
- class LCD
  Class for LCD S2004A display driver.

Macros

- #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.

Variables

- LCD * p_global_BSW_lcd

4.27.1 Detailed Description

LCD class header file.

Date

20 avr. 2019

Author

nicls67

4.27.2 Macro Definition Documentation

4.27.2.1 BACKLIGHT_PIN

```
#define BACKLIGHT_PIN 3
```

Backlight pin is on P3

Definition at line 17 of file LCD.h.

4.27.2.2 EN_PIN

```
#define EN_PIN 2
```

EN bit is on P2

Definition at line 14 of file LCD.h.

4.27.2.3 LCD_CNF_BACKLIGHT_OFF

```
#define LCD_CNF_BACKLIGHT_OFF 0
```

Backlight is disabled

Definition at line 70 of file LCD.h.

4.27.2.4 LCD_CNF_BACKLIGHT_ON

```
#define LCD_CNF_BACKLIGHT_ON 1
```

Backlight is enabled

Definition at line 69 of file LCD.h.

4.27.2.5 LCD_CNF_CURSOR_BLINK_OFF

```
#define LCD_CNF_CURSOR_BLINK_OFF 0
```

Cursor blinking is off, bit is set to 0

Definition at line 58 of file LCD.h.

4.27.2.6 LCD_CNF_CURSOR_BLINK_ON

```
#define LCD_CNF_CURSOR_BLINK_ON 1
```

Cursor blinking is on, bit is set to 1

Definition at line 57 of file LCD.h.

4.27.2.7 LCD_CNF_CURSOR_OFF

```
#define LCD_CNF_CURSOR_OFF 0
```

Cursor is off, bit is set to 0

Definition at line 54 of file LCD.h.

4.27.2.8 LCD_CNF_CURSOR_ON

```
#define LCD_CNF_CURSOR_ON 1
```

Cursor is on, bit is set to 1

Definition at line 53 of file LCD.h.

4.27.2.9 LCD_CNF_DISPLAY_OFF

```
#define LCD_CNF_DISPLAY_OFF 0
```

Display is off, bit is set to 0

Definition at line 50 of file LCD.h.

4.27.2.10 LCD_CNF_DISPLAY_ON

```
#define LCD_CNF_DISPLAY_ON 1
```

Display is on, bit is set to 1

Definition at line 49 of file LCD.h.

4.27.2.11 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 62 of file LCD.h.

4.27.2.12 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 61 of file LCD.h.

4.27.2.13 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 66 of file LCD.h.

4.27.2.14 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 65 of file LCD.h.

4.27.2.15 LCD_CNF_FONT_5_11

```
#define LCD_CNF_FONT_5_11 1
```

Two-line configuration, bit is set to 1

Definition at line 46 of file LCD.h.

4.27.2.16 LCD_CNF_FONT_5_8

```
#define LCD_CNF_FONT_5_8 0
```

One-line configuration, bit is set to 0

Definition at line 45 of file LCD.h.

4.27.2.17 LCD_CNF_ONE_LINE

```
#define LCD_CNF_ONE_LINE 0
```

One-line configuration, bit is set to 0

Definition at line 41 of file LCD.h.

4.27.2.18 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 37 of file LCD.h.

4.27.2.19 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 38 of file LCD.h.

4.27.2.20 LCD_CNF_TWO_LINE

```
#define LCD_CNF_TWO_LINE 1
```

Two-line configuration, bit is set to 1

Definition at line 42 of file LCD.h.

4.27.2.21 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 34 of file LCD.h.

4.27.2.22 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 33 of file LCD.h.

4.27.2.23 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 32 of file LCD.h.

4.27.2.24 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 27 of file LCD.h.

4.27.2.25 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 29 of file LCD.h.

4.27.2.26 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 28 of file LCD.h.

4.27.2.27 LCD_INST_CLR_DISPLAY_BIT

```
#define LCD_INST_CLR_DISPLAY_BIT 0
```

Instruction bit for "clear display" is DB0

Definition at line 20 of file LCD.h.

4.27.2.28 LCD_INST_DISPLAY_CTRL

```
#define LCD_INST_DISPLAY_CTRL 3
```

Instruction bit for "display control" is DB3

Definition at line 22 of file LCD.h.

4.27.2.29 LCD_INST_ENTRY_MODE_SET

```
#define LCD_INST_ENTRY_MODE_SET 2
```

Instruction bit for "entry mode" is DB2

Definition at line 23 of file LCD.h.

4.27.2.30 LCD_INST_FUNCTION_SET

```
#define LCD_INST_FUNCTION_SET 5
```

Instruction bit for "function set" is DB5

Definition at line 21 of file LCD.h.

4.27.2.31 LCD_INST_SET_DDRAM_ADDR

```
#define LCD_INST_SET_DDRAM_ADDR 7
```

Instruction bit for "set DDRAM address" is DB7

Definition at line 24 of file LCD.h.

4.27.2.32 LCD_RAM_1_LINE_MAX

```
#define LCD_RAM_1_LINE_MAX 0x4F
```

Maximum address value in 1-line mode

Definition at line 74 of file LCD.h.

4.27.2.33 LCD_RAM_1_LINE_MIN

```
#define LCD_RAM_1_LINE_MIN 0
```

Minimum address value in 1-line mode

Definition at line 73 of file LCD.h.

4.27.2.34 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 76 of file LCD.h.

4.27.2.35 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 78 of file LCD.h.

4.27.2.36 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 75 of file LCD.h.

4.27.2.37 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 77 of file LCD.h.

4.27.2.38 LCD_SIZE_NB_CHAR_PER_LINE

```
#define LCD_SIZE_NB_CHAR_PER_LINE 20
```

LCD screen has 20 characters per line

Definition at line 85 of file LCD.h.

4.27.2.39 LCD_SIZE_NB_LINES

```
#define LCD_SIZE_NB_LINES 4
```

LCD screen has 4 lines

Definition at line 86 of file LCD.h.

4.27.2.40 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 81 of file LCD.h.

4.27.2.41 LCD_WAIT_OTHER_MODES

```
#define LCD_WAIT_OTHER_MODES 40
```

Waiting time after all other modes is at least 38 us

Definition at line 82 of file LCD.h.

4.27.2.42 RS_PIN

```
#define RS_PIN 0
```

RS pin is on P0

Definition at line 16 of file LCD.h.

4.27.2.43 RW_PIN

```
#define RW_PIN 1
```

RW pin is on P1

Definition at line 15 of file LCD.h.

4.27.3 Enumeration Type Documentation

4.27.3.1 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 93 of file LCD.h.

4.27.3.2 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 107 of file LCD.h.

4.27.3.3 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 118 of file LCD.h.

4.27.4 Variable Documentation

4.27.4.1 p_global_BSW_lcd

```
LCD* p_global_BSW_lcd
```

Pointer to LCD driver object

Definition at line 16 of file LCD.cpp.

4.28 LinkedList.cpp File Reference

Linked List library source file.

```
#include <stdlib.h>
#include <avr/io.h>
#include "../string/String.h"
#include "../operators/operators.h"
#include "../../bsw/usart/usart.h"
#include "../../bsw/timer/timer.h"
#include "../../bsw/I2C/I2C.h"
#include "../../bsw/lcd/LCD.h"
#include "../../bsw/dio/dio.h"
#include "../../bsw/dht22/dht22.h"
#include "../../bsw/cpuLoad/CpuLoad.h"
#include "../../bsw/bsw.h"
#include "../../asw/TempSensor/TempSensor.h"
#include "../../asw/debug_ift/DebugInterface.h"
#include "../../asw/debug_mgt/DebugManagement.h"
#include "../../asw/display_ift/DisplayInterface.h"
#include "../../asw/keepAliveLed/keepAliveLed.h"
#include "../../asw/display_mgt/DisplayManagement.h"
#include "../../asw/asw.h"
#include "LinkedList.h"
```

Include dependency graph for LinkedList.cpp:

4.28.1 Detailed Description

Linked List library source file.

Date

27 avr. 2019

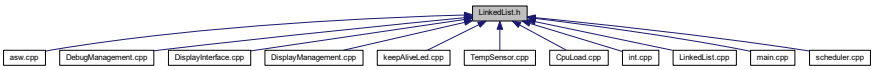
Author

nicls67

4.29 LinkedList.h File Reference

Linked List library header file.

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



Classes

- class LinkedList

  Linked list class.
- struct LinkedList::T_LL_element

  Type defining a linked list element.

Typedefs

- typedef bool(∗ CompareFctPtr_t) (void ∗LLElement, void ∗CompareElement)

4.29.1 Detailed Description

Linked List library header file.

Date

27 avr. 2019

Author

nicls67

4.29.2 Typedef Documentation

4.29.2.1 CompareFctPtr_t

```
typedef bool(* CompareFctPtr_t) (void *LLElement, void *CompareElement)
```

Type defining a pointer to the comparison function

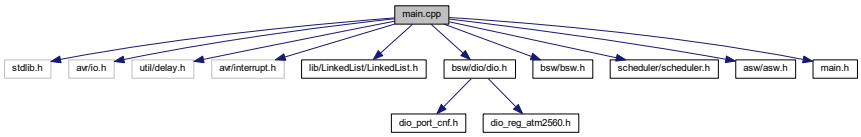
Definition at line 14 of file LinkedList.h.

4.30 main.cpp File Reference

Background task file.

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

Include dependency graph for main.cpp:



Macros

- #define DEBUG_ACTIVE_PORT ENCODE_PORT(PORT_B, 4)

Functions

- int main (void)

  Background task of program.

Variables

- bool isDebugModeActivated
- const T_ASW_init_cnf ASW_init_cnf

4.30.1 Detailed Description

Background task file.

Date

12 mars 2018

Author

nicls67

4.30.2 Macro Definition Documentation

4.30.2.1 DEBUG_ACTIVE_PORT

```
#define DEBUG_ACTIVE_PORT ENCODE_PORT(PORT_B, 4)
```

Debug activation pin is port PB6

Definition at line 26 of file main.cpp.

4.30.3 Function Documentation

4.30.3.1 main()

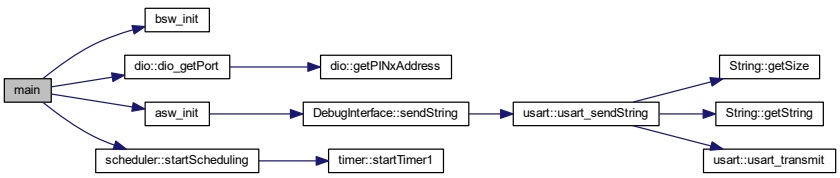
```
int main (
            void  )
```

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 45 of file main.cpp.

Here is the call graph for this function:



4.30.4 Variable Documentation

4.30.4.1 ASW_init_cnf

```
const T_ASW_init_cnf ASW_init_cnf
```

Initial value:

```
=
{
    true,
    true,
    true,
    true
}
```

Definition of needed ASW services

Definition at line 30 of file main.cpp.

4.30.4.2 isDebugModeActivated

```
bool isDebugModeActivated
```

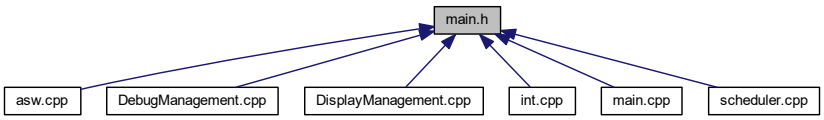
Flag indicating if the debug mode is activated or not

Definition at line 28 of file main.cpp.

4.31 main.h File Reference

Background task header file.

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



Variables

- bool isDebugModeActivated
- const T_ASW_init_cnf ASW_init_cnf

4.31.1 Detailed Description

Background task header file.

Date

17 mars 2018

Author

nicls67

4.31.2 Variable Documentation

4.31.2.1 ASW_init_cnf

```
const T_ASW_init_cnf ASW_init_cnf
```

Definition of needed ASW services

Definition at line 30 of file main.cpp.

4.31.2.2 isDebugModeActivated

```
bool isDebugModeActivated
```

Flag indicating if the debug mode is activated or not

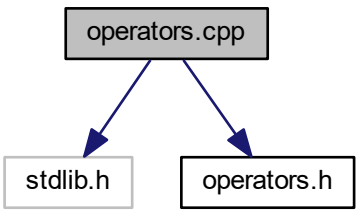
Definition at line 28 of file main.cpp.

4.32 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.

4.32.1 Detailed Description

c++ operators definitions

Date

14 mars 2018

Author

nicls67

4.32.2 Function Documentation

4.32.2.1 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.

4.32.2.2 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

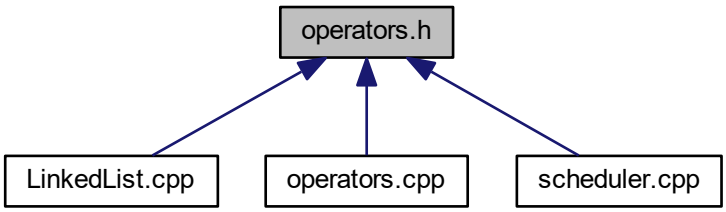
Pointer to the start of allocated memory zone

Definition at line 13 of file operators.cpp.

4.33 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.

4.33.1 Detailed Description

c++ operators definitions header file

Date

14 mars 2018

Author

nicls67

4.33.2 Function Documentation

4.33.2.1 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.

4.33.2.2 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

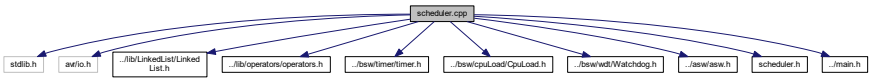
Definition at line 13 of file operators.cpp.

4.34 scheduler.cpp File Reference

Defines scheduler class.

```
#include <stdlib.h>
#include <avr/io.h>
#include "../lib/LinkedList/LinkedList.h"
#include "../lib/operators/operators.h"
#include "../bsw/timer/timer.h"
#include "../bsw/cpuLoad/CpuLoad.h"
#include "../bsw/wdt/Watchdog.h"
#include "../asw/asw.h"
#include "scheduler.h"
#include "../main.h"
```

Include dependency graph for scheduler.cpp:



Variables

- scheduler * p_global_scheduler

4.34.1 Detailed Description

Defines scheduler class.

Date

16 mars 2018

Author

nicls67

4.34.2 Variable Documentation

4.34.2.1 p_global_scheduler

```
scheduler* p_global_scheduler
```

Pointer to scheduler object

Definition at line 27 of file scheduler.cpp.

4.35 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.

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_global_scheduler

4.35.1 Detailed Description

Scheduler class header file.

Date

16 mars 2018

Author

nicls67

4.35.2 Macro Definition Documentation

4.35.2.1 PRESCALER_PERIODIC_TIMER

```
#define PRESCALER_PERIODIC_TIMER 256
```

Value of prescaler to use for periodic timer

Definition at line 16 of file scheduler.h.

4.35.2.2 SW_PERIOD_MS

```
#define SW_PERIOD_MS 500
```

Software period, used to define periodic timer interrupt

Definition at line 15 of file scheduler.h.

4.35.2.3 TIMER_CTC_VALUE

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

Compare value for periodic timer

Definition at line 17 of file scheduler.h.

4.35.3 Typedef Documentation

4.35.3.1 TaskPtr_t

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

Type defining a pointer to function.

Definition at line 22 of file scheduler.h.

4.35.4 Variable Documentation

4.35.4.1 p_global_scheduler

```
scheduler* p_global_scheduler
```

Pointer to scheduler object

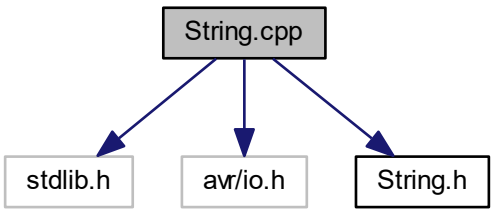
Definition at line 27 of file scheduler.cpp.

4.36 String.cpp File Reference

String class source file.

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

Include dependency graph for String.cpp:



4.36.1 Detailed Description

String class source file.

Date

2 mai 2019

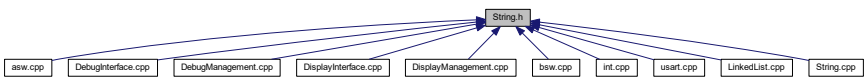
Author

nicls67

4.37 String.h File Reference

String class header file.

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



Classes

- class String

  String management class.

4.37.1 Detailed Description

String class header file.

Date

2 mai 2019

Author

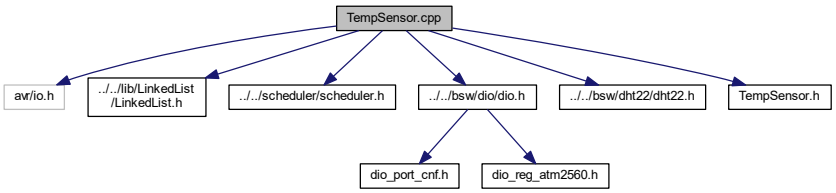
nicls67

4.38 TempSensor.cpp File Reference

Defines function of class TempSensor.

```
#include <avr/io.h>
#include "../../lib/LinkedList/LinkedList.h"
#include "../../scheduler/scheduler.h"
#include "../../bsw/dio/dio.h"
#include "../../bsw/dht22/dht22.h"
#include "TempSensor.h"
```

Include dependency graph for TempSensor.cpp:



Macros

- #define PIT_BEFORE_INVALID 60

Variables

- TempSensor * p_global_ASW_TempSensor

4.38.1 Detailed Description

Defines function of class TempSensor.

Date

23 mars 2018

Author

nicls67

4.38.2 Macro Definition Documentation

4.38.2.1 PIT_BEFORE_INVALID

```
#define PIT_BEFORE_INVALID 60
```

Definition at line 20 of file TempSensor.cpp.

4.38.3 Variable Documentation

4.38.3.1 p_global_ASW_TempSensor

```
TempSensor* p_global_ASW_TempSensor
```

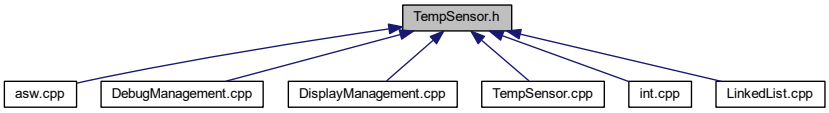
Pointer to TempSensor object

Definition at line 22 of file TempSensor.cpp.

4.39 TempSensor.h File Reference

Class TempSensor header file.

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
- #define DHT22_PORT ENCODE_PORT(PORT_B, 6)

Variables

- TempSensor * p_global_ASW_TempSensor

4.39.1 Detailed Description

Class TempSensor header file.

Date

23 mars 2018

Author

nicls67

4.39.2 Macro Definition Documentation

4.39.2.1 DHT22_PORT

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

DHT22 is connected to port PB6

Definition at line 15 of file TempSensor.h.

4.39.2.2 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.

4.39.3 Variable Documentation

4.39.3.1 p_global_ASW_TempSensor

```
TempSensor* p_global_ASW_TempSensor
```

Pointer to TempSensor object

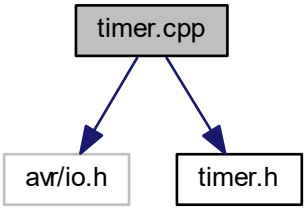
Definition at line 22 of file TempSensor.cpp.

4.40 timer.cpp File Reference

Defines function for class timer.

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

Include dependency graph for timer.cpp:



Variables

- timer * p_global_BSW_timer

4.40.1 Detailed Description

Defines function for class timer.

Date

15 mars 2018

Author

nicls67

4.40.2 Variable Documentation

4.40.2.1 p_global_BSW_timer

```
timer* p_global_BSW_timer
```

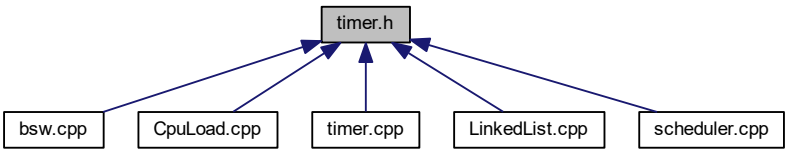
Pointer to timer driver object

Definition at line 13 of file timer.cpp.

4.41 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.

Variables

- timer * p_global_BSW_timer

4.41.1 Detailed Description

Timer class header file.

Date

15 mars 2018

Author

nicls67

4.41.2 Variable Documentation

4.41.2.1 p_global_BSW_timer

```
timer* p_global_BSW_timer
```

Pointer to timer driver object

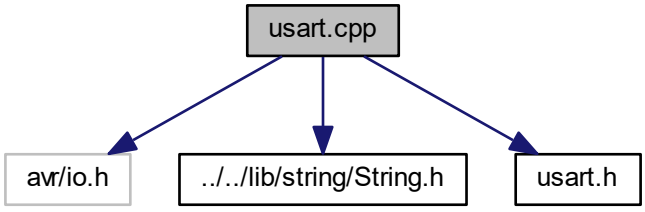
Definition at line 13 of file timer.cpp.

4.42 usart.cpp File Reference

BSW library for USART.

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

Include dependency graph for usart.cpp:



Variables

- usart * p_global_BSW_usart

4.42.1 Detailed Description

BSW library for USART.

Date

13 mars 2018

Author

nicls67

4.42.2 Variable Documentation

4.42.2.1 p_global_BSW_usart

```
usart* p_global_BSW_usart
```

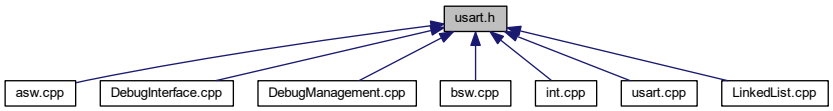
Pointer to usart driver object

Definition at line 16 of file usart.cpp.

4.43 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.

Variables

- usart * p_global_BSW_usart

4.43.1 Detailed Description

Header file for USART library.

Date

13 mars 2018

Author

nicls67

4.43.2 Variable Documentation

4.43.2.1 p_global_BSW_usart

```
usart* p_global_BSW_usart
```

Pointer to usart driver object

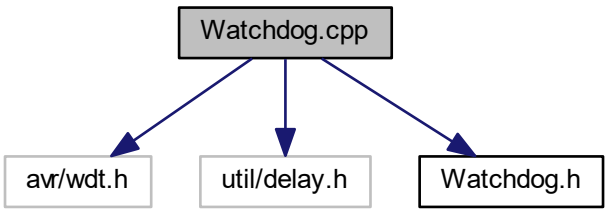
Definition at line 16 of file usart.cpp.

4.44 Watchdog.cpp File Reference

Class Watchdog source code file.

```
#include <avr/wdt.h>
#include <util/delay.h>
#include "Watchdog.h"
```

Include dependency graph for Watchdog.cpp:



Macros

- #define WDG_TIMEOUT_DEFAULT_MS WDG_TMO_500MS

Variables

- Watchdog * p_global_BSW_wdg

4.44.1 Detailed Description

Class Watchdog source code file.

Date

6 juin 2019

Author

nicls67

4.44.2 Macro Definition Documentation

4.44.2.1 WDG_TIMEOUT_DEFAULT_MS

```
#define WDG_TIMEOUT_DEFAULT_MS WDG_TMO_500MS
```

Default timeout value is set to 500 ms

Definition at line 19 of file Watchdog.cpp.

4.44.3 Variable Documentation

4.44.3.1 p_global_BSW_wdg

```
Watchdog* p_global_BSW_wdg
```

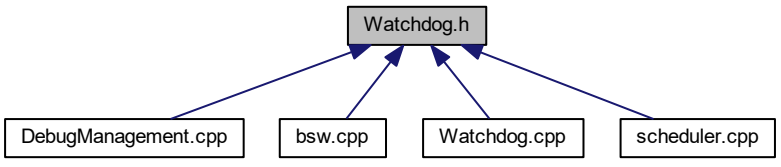
Pointer to Watchdog driver object

Definition at line 21 of file Watchdog.cpp.

4.45 Watchdog.h File Reference

Class Watchdog header file.

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



Classes

- class Watchdog

  Watchdog management class.

Macros

- #define WDG_TMO_15MS WDTO_15MS

  Definition of available timeout values.
- #define WDG_TMO_30MS WDTO_30MS
- #define WDG_TMO_60MS WDTO_60MS
- #define WDG_TMO_120MS WDTO_120MS
- #define WDG_TMO_250MS WDTO_250MS
- #define WDG_TMO_500MS WDTO_500MS
- #define WDG_TMO_1S WDTO_1S
- #define WDG_TMO_2S WDTO_2S
- #define WDG_TMO_4S WDTO_4S
- #define WDG_TMO_8S WDTO_8S

Variables

- Watchdog * p_global_BSW_wdg

4.45.1 Detailed Description

Class Watchdog header file.

Date

6 juin 2019

Author

nicls67

4.45.2 Macro Definition Documentation

4.45.2.1 WDG_TMO_120MS

```
#define WDG_TMO_120MS WDTO_120MS
```

Timeout value is 120 ms

Definition at line 19 of file Watchdog.h.

4.45.2.2 WDG_TMO_15MS

```
#define WDG_TMO_15MS WDTO_15MS
```

Definition of available timeout values.

Timeout value is 15 ms

Definition at line 16 of file Watchdog.h.

4.45.2.3 WDG_TMO_1S

```
#define WDG_TMO_1S WDTO_1S
```

Timeout value is 1 s

Definition at line 22 of file Watchdog.h.

4.45.2.4 WDG_TMO_250MS

```
#define WDG_TMO_250MS WDTO_250MS
```

Timeout value is 250 ms

Definition at line 20 of file Watchdog.h.

4.45.2.5 WDG_TMO_2S

```
#define WDG_TMO_2S WDTO_2S
```

Timeout value is 2 s

Definition at line 23 of file Watchdog.h.

4.45.2.6 WDG_TMO_30MS

```
#define WDG_TMO_30MS WDTO_30MS
```

Timeout value is 30 ms

Definition at line 17 of file Watchdog.h.

4.45.2.7 WDG_TMO_4S

```
#define WDG_TMO_4S WDTO_4S
```

Timeout value is 4 s

Definition at line 24 of file Watchdog.h.

4.45.2.8 WDG_TMO_500MS

```
#define WDG_TMO_500MS WDTO_500MS
```

Timeout value is 500 ms

Definition at line 21 of file Watchdog.h.

4.45.2.9 WDG_TMO_60MS

```
#define WDG_TMO_60MS WDTO_60MS
```

Timeout value is 60 ms

Definition at line 18 of file Watchdog.h.

4.45.2.10 WDG_TMO_8S

```
#define WDG_TMO_8S WDTO_8S
```

Timeout value is 8 s

Definition at line 25 of file Watchdog.h.

4.45.3 Variable Documentation

4.45.3.1 p_global_BSW_wdg

```
Watchdog* p_global_BSW_wdg
```

Pointer to Watchdog driver object

Definition at line 21 of file Watchdog.cpp.

Index