

Arduino

1.0

Generated by Doxygen 1.8.14

Contents

1	Class Index	1
1.1	Class List	1
2	File Index	3
2.1	File List	3
3	Class Documentation	5
3.1	CpuLoad Class Reference	5
3.1.1	Detailed Description	5
3.1.2	Constructor & Destructor Documentation	6
3.1.2.1	CpuLoad()	6
3.1.3	Member Function Documentation	6
3.1.3.1	ComputeCPULoad()	6
3.1.3.2	getAverageCPULoad()	7
3.1.3.3	getCurrentCPULoad()	7
3.1.3.4	getMaxCPULoad()	8
3.1.4	Member Data Documentation	8
3.1.4.1	avg_load	8
3.1.4.2	current_load	8
3.1.4.3	last_sum_value	9
3.1.4.4	max_load	9
3.1.4.5	sample_cnt	9
3.1.4.6	sample_idx	9
3.1.4.7	sample_mem	9

3.2	dht22 Class Reference	10
3.2.1	Detailed Description	10
3.2.2	Constructor & Destructor Documentation	10
3.2.2.1	<code>dht22()</code>	10
3.2.3	Member Function Documentation	11
3.2.3.1	<code>initializeCommunication()</code>	11
3.2.3.2	<code>read()</code>	11
3.3	dio Class Reference	12
3.3.1	Detailed Description	13
3.3.2	Constructor & Destructor Documentation	13
3.3.2.1	<code>dio()</code>	14
3.3.3	Member Function Documentation	14
3.3.3.1	<code>dio_changePortPinCnf()</code>	14
3.3.3.2	<code>dio_getPort()</code>	15
3.3.3.3	<code>dio_getPort_fast()</code>	16
3.3.3.4	<code>dio_invertPort()</code>	16
3.3.3.5	<code>dio_memorizePINaddress()</code>	17
3.3.3.6	<code>dio_setPort()</code>	18
3.3.3.7	<code>getDDRxAddress()</code>	19
3.3.3.8	<code>getPINxAddress()</code>	20
3.3.3.9	<code>getPORTxAddress()</code>	20
3.3.3.10	<code>ports_init()</code>	21
3.3.4	Member Data Documentation	21
3.3.4.1	<code>PINx_addr_mem</code>	21
3.3.4.2	<code>PINx_idx_mem</code>	22
3.4	DisplayInterface Class Reference	22
3.4.1	Detailed Description	23
3.4.2	Constructor & Destructor Documentation	23
3.4.2.1	<code>DisplayInterface()</code>	23
3.4.3	Member Function Documentation	24

3.4.3.1	ClearLine()	24
3.4.3.2	DisplayFullLine()	25
3.4.3.3	FindFirstCharAddr()	26
3.4.3.4	getLLShiftDataPtr()	26
3.4.3.5	IsLineEmpty()	27
3.4.3.6	LLElementCompare()	27
3.4.3.7	shiftLine_task()	28
3.4.4	Member Data Documentation	29
3.4.4.1	dummy	29
3.4.4.2	lineEmptyTab	29
3.4.4.3	LL_shift_data_ptr	29
3.4.4.4	p_lcd	29
3.5	I2C Class Reference	30
3.5.1	Detailed Description	30
3.5.2	Constructor & Destructor Documentation	30
3.5.2.1	I2C()	30
3.5.3	Member Function Documentation	31
3.5.3.1	initializeBus()	31
3.5.3.2	setBitRate()	32
3.5.3.3	setTxAddress()	32
3.5.3.4	writeByte()	33
3.5.4	Member Data Documentation	33
3.5.4.1	bitrate	33
3.5.4.2	tx_address	33
3.6	keepAliveLed Class Reference	34
3.6.1	Detailed Description	34
3.6.2	Constructor & Destructor Documentation	34
3.6.2.1	keepAliveLed()	34
3.6.3	Member Function Documentation	35
3.6.3.1	blinkLed_task()	35

3.7	LCD Class Reference	35
3.7.1	Detailed Description	37
3.7.2	Constructor & Destructor Documentation	37
3.7.2.1	LCD()	37
3.7.3	Member Function Documentation	38
3.7.3.1	command()	38
3.7.3.2	ConfigureBacklight()	38
3.7.3.3	ConfigureCursorBlink()	39
3.7.3.4	ConfigureCursorOnOff()	40
3.7.3.5	ConfigureDisplayOnOff()	41
3.7.3.6	ConfigureEntryModeDir()	41
3.7.3.7	ConfigureEntryModeShift()	42
3.7.3.8	ConfigureFontType()	42
3.7.3.9	ConfigureI2CAddr()	43
3.7.3.10	ConfigureLineNumber()	44
3.7.3.11	GetDDRAMAddress()	45
3.7.3.12	GetLineNumberCnf()	45
3.7.3.13	InitializeScreen()	46
3.7.3.14	SetDDRAMAddress()	46
3.7.3.15	write()	47
3.7.3.16	write4bits()	48
3.7.3.17	WriteInRam()	49
3.7.4	Member Data Documentation	50
3.7.4.1	backlight_enable	50
3.7.4.2	cnfCursorBlink	50
3.7.4.3	cnfCursorOnOff	50
3.7.4.4	cnfDisplayOnOff	50
3.7.4.5	cnfEntryModeDir	50
3.7.4.6	cnfEntryModeShift	51
3.7.4.7	cnfFontType	51

3.7.4.8	<code>cnfI2C_addr</code>	51
3.7.4.9	<code>cnfLineNumber</code>	51
3.7.4.10	<code>ddram_addr</code>	51
3.8	<code>LinkedList</code> Class Reference	52
3.8.1	Detailed Description	53
3.8.2	Member Typedef Documentation	53
3.8.2.1	<code>T_LL_element</code>	53
3.8.3	Constructor & Destructor Documentation	53
3.8.3.1	<code>LinkedList()</code>	53
3.8.4	Member Function Documentation	53
3.8.4.1	<code>AttachNewElement()</code>	53
3.8.4.2	<code>getCurrentElement()</code>	54
3.8.4.3	<code>IsLLEmpty()</code>	55
3.8.4.4	<code>MoveToNextElement()</code>	55
3.8.4.5	<code>RemoveElement()</code>	56
3.8.4.6	<code>ResetElementPtr()</code>	57
3.8.5	Member Data Documentation	57
3.8.5.1	<code>curElement_ptr</code>	58
3.8.5.2	<code>firstElement</code>	58
3.9	<code>scheduler</code> Class Reference	58
3.9.1	Detailed Description	59
3.9.2	Member Typedef Documentation	59
3.9.2.1	<code>Task_t</code>	60
3.9.3	Constructor & Destructor Documentation	60
3.9.3.1	<code>scheduler()</code>	60
3.9.4	Member Function Documentation	60
3.9.4.1	<code>addPeriodicTask()</code>	60
3.9.4.2	<code>getPitNumber()</code>	61
3.9.4.3	<code>launchPeriodicTasks()</code>	62
3.9.4.4	<code>LLElementCompare()</code>	62

3.9.4.5	removePeriodicTask()	63
3.9.4.6	startScheduling()	64
3.9.5	Member Data Documentation	65
3.9.5.1	pit_number	65
3.9.5.2	TasksLL_ptr	65
3.10	T_ASW_cnf_struct Struct Reference	66
3.10.1	Detailed Description	66
3.10.2	Member Data Documentation	66
3.10.2.1	p_DisplayInterface	67
3.10.2.2	p_keepAliveLed	67
3.10.2.3	p_TempSensor	67
3.10.2.4	p_usartDebug	67
3.11	T_BSW_cnf_struct Struct Reference	68
3.11.1	Detailed Description	68
3.11.2	Member Data Documentation	68
3.11.2.1	p_cpupload	68
3.11.2.2	p_dht22	69
3.11.2.3	p_dio	69
3.11.2.4	p_i2c	69
3.11.2.5	p_lcd	69
3.11.2.6	p_timer	69
3.11.2.7	p_usart	70
3.12	T_Display_shift_data Struct Reference	70
3.12.1	Detailed Description	70
3.12.2	Member Data Documentation	70
3.12.2.1	line	70
3.12.2.2	size	71
3.12.2.3	str_cur_ptr	71
3.12.2.4	str_start_ptr	71
3.12.2.5	temporization	71

3.13 T_LCD_conf_struct Struct Reference	71
3.13.1 Detailed Description	72
3.13.2 Member Data Documentation	72
3.13.2.1 backlight_en	72
3.13.2.2 cursor_en	72
3.13.2.3 cursorBlink_en	72
3.13.2.4 display_en	72
3.13.2.5 entryModeDir	72
3.13.2.6 entryModeShift	73
3.13.2.7 fontType_cnf	73
3.13.2.8 i2c_addr	73
3.13.2.9 lineNumber_cnf	73
3.14 LinkedList::T_LL_element Struct Reference	73
3.14.1 Detailed Description	74
3.14.2 Member Data Documentation	74
3.14.2.1 data_ptr	74
3.14.2.2 nextElement	74
3.15 scheduler::Task_t Struct Reference	74
3.15.1 Detailed Description	75
3.15.2 Member Data Documentation	75
3.15.2.1 period	75
3.15.2.2 TaskPtr	75
3.16 TempSensor Class Reference	75
3.16.1 Detailed Description	76
3.16.2 Constructor & Destructor Documentation	76
3.16.2.1 TempSensor()	77
3.16.3 Member Function Documentation	77
3.16.3.1 getHumidity()	77
3.16.3.2 getHumPtr()	78
3.16.3.3 getTemp()	78

3.16.3.4	getTempPtr()	79
3.16.3.5	readTempSensor_task()	79
3.16.3.6	setValidity()	80
3.16.3.7	updateLastValidValues()	81
3.16.4	Member Data Documentation	81
3.16.4.1	read_humidity	81
3.16.4.2	read_temperature	82
3.16.4.3	valid_hum	82
3.16.4.4	valid_pit	82
3.16.4.5	valid_temp	82
3.16.4.6	validity	82
3.16.4.7	validity_last_read	83
3.17	timer Class Reference	83
3.17.1	Detailed Description	83
3.17.2	Constructor & Destructor Documentation	83
3.17.2.1	timer()	84
3.17.3	Member Function Documentation	84
3.17.3.1	configureTimer1()	84
3.17.3.2	getTimer1Value()	85
3.17.3.3	startTimer1()	85
3.17.3.4	stopTimer1()	86
3.17.4	Member Data Documentation	86
3.17.4.1	prescaler	86
3.18	uart Class Reference	86
3.18.1	Detailed Description	87
3.18.2	Constructor & Destructor Documentation	87
3.18.2.1	uart()	87
3.18.3	Member Function Documentation	88
3.18.3.1	setBaudRate()	88
3.18.3.2	uart_init()	88

3.18.3.3 uart_read()	89
3.18.3.4 uart_sendString()	89
3.18.3.5 uart_transmit()	90
3.18.4 Member Data Documentation	91
3.18.4.1 BaudRate	91
3.19 UsartDebug Class Reference	91
3.19.1 Detailed Description	92
3.19.2 Constructor & Destructor Documentation	92
3.19.2.1 UsartDebug()	92
3.19.3 Member Function Documentation	92
3.19.3.1 activateDebugMode()	93
3.19.3.2 DebugModeManagement()	93
3.19.3.3 DisplayCPULoad_task()	94
3.19.3.4 DisplaySensors_task()	95
3.19.3.5 isDebugModeActive()	96
3.19.3.6 sendBool()	96
3.19.3.7 sendData()	97
3.19.3.8 sendInteger()	98
3.19.4 Member Data Documentation	99
3.19.4.1 debug_state	99
3.19.4.2 debugModeActive_F	99

4 File Documentation	101
4.1 asw.cpp File Reference	101
4.1.1 Detailed Description	102
4.1.2 Function Documentation	102
4.1.2.1 asw_init()	102
4.1.3 Variable Documentation	102
4.1.3.1 ASW_cnf_struct	103
4.2 asw.h File Reference	103
4.2.1 Detailed Description	104
4.2.2 Function Documentation	104
4.2.2.1 asw_init()	104
4.2.3 Variable Documentation	105
4.2.3.1 ASW_cnf_struct	105
4.3 bsw.cpp File Reference	105
4.3.1 Detailed Description	106
4.3.2 Function Documentation	106
4.3.2.1 bsw_init()	106
4.3.3 Variable Documentation	106
4.3.3.1 BSW_cnf_struct	107
4.4 bsw.h File Reference	107
4.4.1 Detailed Description	108
4.4.2 Macro Definition Documentation	108
4.4.2.1 I2C_BITRATE	108
4.4.2.2 USART_BAUDRATE	109
4.4.3 Function Documentation	109
4.4.3.1 bsw_init()	109
4.4.4 Variable Documentation	109
4.4.4.1 BSW_cnf_struct	109
4.5 CpuLoad.cpp File Reference	110
4.5.1 Detailed Description	110

4.6 CpuLoad.h File Reference	110
4.6.1 Detailed Description	112
4.6.2 Macro Definition Documentation	112
4.6.2.1 NB_OF_SAMPLES	112
4.7 debug.cpp File Reference	112
4.7.1 Detailed Description	113
4.7.2 Variable Documentation	113
4.7.2.1 str_debug_main_menu	113
4.8 debug.h File Reference	114
4.8.1 Detailed Description	115
4.8.2 Macro Definition Documentation	115
4.8.2.1 PERIOD_MS_TASK_DISPLAY_CPU_LOAD	115
4.8.2.2 PERIOD_MS_TASK_DISPLAY_SENSORS	115
4.8.3 Enumeration Type Documentation	115
4.8.3.1 debug_state_t	115
4.9 dht22.cpp File Reference	116
4.9.1 Detailed Description	116
4.9.2 Macro Definition Documentation	117
4.9.2.1 MAX_WAIT_TIME_US	117
4.10 dht22.h File Reference	117
4.10.1 Detailed Description	118
4.10.2 Macro Definition Documentation	118
4.10.2.1 DHT22_PORT	118
4.11 dio.cpp File Reference	118
4.11.1 Detailed Description	119
4.12 dio.h File Reference	119
4.12.1 Detailed Description	120
4.12.2 Macro Definition Documentation	120
4.12.2.1 DECODE_PIN	120
4.12.2.2 DECODE_PORT	121

4.12.2.3 ENCODE_PORT	121
4.12.2.4 PORT_A	121
4.12.2.5 PORT_B	121
4.12.2.6 PORT_C	121
4.12.2.7 PORT_CNF_IN	122
4.12.2.8 PORT_CNF_OUT	122
4.12.2.9 PORT_D	122
4.13 dio_port_cnf.h File Reference	122
4.13.1 Detailed Description	123
4.13.2 Macro Definition Documentation	123
4.13.2.1 PORTB_CNF_DDRB	123
4.13.2.2 PORTB_CNF_PORTB	124
4.14 dio_reg_atm2560.h File Reference	124
4.14.1 Macro Definition Documentation	125
4.14.1.1 DDRA_PTR	125
4.14.1.2 DDRB_PTR	125
4.14.1.3 DDRC_PTR	125
4.14.1.4 DDRD_PTR	125
4.14.1.5 PINA_PTR	126
4.14.1.6 PINB_PTR	126
4.14.1.7 PINC_PTR	126
4.14.1.8 PIND_PTR	126
4.14.1.9 PORTA_PTR	126
4.14.1.10 PORTB_PTR	127
4.14.1.11 PORTC_PTR	127
4.14.1.12 PORTD_PTR	127
4.15 DisplayInterface.cpp File Reference	127
4.15.1 Detailed Description	128
4.16 DisplayInterface.h File Reference	128
4.16.1 Detailed Description	130

4.16.2 Macro Definition Documentation	130
4.16.2.1 DISPLAY_LINE_SHIFT_PERIOD_MS	130
4.16.2.2 DISPLAY_LINE_SHIFT_TEMPO_TIME	130
4.16.3 Enumeration Type Documentation	130
4.16.3.1 T_DisplayInterface_LineDisplayMode	130
4.16.4 Variable Documentation	131
4.16.4.1 LCD_init_cnf	131
4.17 I2C.cpp File Reference	131
4.17.1 Detailed Description	132
4.18 I2C.h File Reference	132
4.18.1 Detailed Description	134
4.18.2 Macro Definition Documentation	134
4.18.2.1 DATA_ACK	134
4.18.2.2 SLA_ACK	134
4.18.2.3 START	134
4.19 keepAliveLed.cpp File Reference	135
4.19.1 Detailed Description	135
4.20 keepAliveLed.h File Reference	135
4.20.1 Detailed Description	137
4.20.2 Macro Definition Documentation	137
4.20.2.1 LED_PORT	137
4.20.2.2 PERIOD_MS_TASK_LED	137
4.21 LCD.cpp File Reference	137
4.21.1 Detailed Description	138
4.22 LCD.h File Reference	138
4.22.1 Detailed Description	140
4.22.2 Macro Definition Documentation	140
4.22.2.1 BACKLIGHT_PIN	140
4.22.2.2 EN_PIN	140
4.22.2.3 LCD_CNF_BACKLIGHT_OFF	141

4.22.2.4 LCD_CNF_BACKLIGHT_ON	141
4.22.2.5 LCD_CNF_CURSOR_BLINK_OFF	141
4.22.2.6 LCD_CNF_CURSOR_BLINK_ON	141
4.22.2.7 LCD_CNF_CURSOR_OFF	141
4.22.2.8 LCD_CNF_CURSOR_ON	142
4.22.2.9 LCD_CNF_DISPLAY_OFF	142
4.22.2.10 LCD_CNF_DISPLAY_ON	142
4.22.2.11 LCD_CNF_ENTRY_MODE_DIRECTION_LEFT	142
4.22.2.12 LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT	142
4.22.2.13 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF	143
4.22.2.14 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON	143
4.22.2.15 LCD_CNF_FONT_5_11	143
4.22.2.16 LCD_CNF_FONT_5_8	143
4.22.2.17 LCD_CNF_ONE_LINE	143
4.22.2.18 LCD_CNF_SHIFT_ID	144
4.22.2.19 LCD_CNF_SHIFT_SH	144
4.22.2.20 LCD_CNF_TWO_LINE	144
4.22.2.21 LCD_DISPLAY_CTRL_FIELD_B	144
4.22.2.22 LCD_DISPLAY_CTRL_FIELD_C	144
4.22.2.23 LCD_DISPLAY_CTRL_FIELD_D	145
4.22.2.24 LCD_FCT_SET_FIELD_DL	145
4.22.2.25 LCD_FCT_SET_FIELD_F	145
4.22.2.26 LCD_FCT_SET_FIELD_N	145
4.22.2.27 LCD_INST_CLR_DISPLAY_BIT	145
4.22.2.28 LCD_INST_DISPLAY_CTRL	146
4.22.2.29 LCD_INST_ENTRY_MODE_SET	146
4.22.2.30 LCD_INST_FUNCTION_SET	146
4.22.2.31 LCD_INST_SET_DDRAM_ADDR	146
4.22.2.32 LCD_RAM_1_LINE_MAX	146
4.22.2.33 LCD_RAM_1_LINE_MIN	147

4.22.2.34 LCD_RAM_2_LINES_MAX_1	147
4.22.2.35 LCD_RAM_2_LINES_MAX_2	147
4.22.2.36 LCD_RAM_2_LINES_MIN_1	147
4.22.2.37 LCD_RAM_2_LINES_MIN_2	147
4.22.2.38 LCD_SIZE_NB_CHAR_PER_LINE	148
4.22.2.39 LCD_SIZE_NB_LINES	148
4.22.2.40 LCD_WAIT_CLR_RETURN	148
4.22.2.41 LCD_WAIT_OTHER_MODES	148
4.22.2.42 RS_PIN	148
4.22.2.43 RW_PIN	149
4.22.3 Enumeration Type Documentation	149
4.22.3.1 T_LCD_command	149
4.22.3.2 T_LCD_config_mode	149
4.22.3.3 T_LCD_ram_area	150
4.23 LinkedList.cpp File Reference	150
4.24 LinkedList.h File Reference	150
4.24.1 Detailed Description	151
4.24.2 Typedef Documentation	151
4.24.2.1 CompareFctPtr_t	152
4.25 main.cpp File Reference	152
4.25.1 Detailed Description	152
4.25.2 Function Documentation	153
4.25.2.1 ISR() [1/2]	153
4.25.2.2 ISR() [2/2]	153
4.25.2.3 main()	154
4.26 main.h File Reference	155
4.26.1 Detailed Description	155
4.27 operators.cpp File Reference	156
4.27.1 Detailed Description	156
4.27.2 Function Documentation	157

4.27.2.1 operator delete()	157
4.27.2.2 operator new()	157
4.28 operators.h File Reference	157
4.28.1 Detailed Description	158
4.28.2 Function Documentation	158
4.28.2.1 operator delete()	159
4.28.2.2 operator new()	159
4.29 scheduler.cpp File Reference	159
4.29.1 Detailed Description	160
4.29.2 Variable Documentation	160
4.29.2.1 p_scheduler	160
4.30 scheduler.h File Reference	161
4.30.1 Detailed Description	162
4.30.2 Macro Definition Documentation	162
4.30.2.1 PRESCALER_PERIODIC_TIMER	162
4.30.2.2 SW_PERIOD_MS	162
4.30.2.3 TIMER_CTC_VALUE	163
4.30.3 Typedef Documentation	163
4.30.3.1 TaskPtr_t	163
4.30.4 Variable Documentation	163
4.30.4.1 p_scheduler	163
4.31 TempSensor.cpp File Reference	163
4.31.1 Detailed Description	164
4.31.2 Macro Definition Documentation	164
4.31.2.1 PIT_BEFORE_INVALID	164
4.32 TempSensor.h File Reference	164
4.32.1 Detailed Description	165
4.32.2 Macro Definition Documentation	165
4.32.2.1 PERIOD_MS_TASK_TEMP_SENSOR	166
4.33 timer.cpp File Reference	166
4.33.1 Detailed Description	166
4.34 timer.h File Reference	167
4.34.1 Detailed Description	167
4.35 usart.cpp File Reference	168
4.35.1 Detailed Description	168
4.36 usart.h File Reference	168
4.36.1 Detailed Description	169
Index	171

Chapter 1

Class Index

1.1 Class List

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

CpuLoad	Class defining CPU load libraries	5
dht22	DHT 22 driver class	10
dio	DIO class	12
DisplayInterface	Display interface services class	22
I2C	Two-wire serial interface (I2C) class definition	30
keepAliveLed	Class for keep-alive LED blinking	34
LCD	Class for LCD S2004A display driver	35
LinkedList	Linked list class	52
scheduler	Scheduler class	58
T_ASW_cnf_struct	ASW configuration structure	66
T_BSW_cnf_struct	BSW configuration structure	68
T_Display_shift_data	Structure containing shift data	70
T_LCD_conf_struct	71
LinkedList::T_LL_element	Type defining a linked list element	73
scheduler::Task_t	Type defining a task structure	74
TempSensor	Class for temperature sensor	75
timer	Class defining a timer	83
usart	USART serial bus class	86
UsartDebug	Class used for debugging on usart link	91

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

asw.cpp	ASW main file	101
asw.h	ASW main header file	103
bsw.cpp	BSW main file	105
bsw.h	BSW main header file	107
CpuLoad.cpp	Defines functions of class CpuLoad	110
CpuLoad.h	CpuLoad class header file	110
debug.cpp	This file defines classes for log and debug data transmission on USART link	112
debug.h	Header file for debug and logging functions	114
dht22.cpp	This file defines classes for DHT22 driver	116
dht22.h	DHT22 driver header file	117
dio.cpp	DIO library	118
dio.h	DIO library header file	119
dio_port_cnf.h	Digital ports configuration file	122
dio_reg_atm2560.h		124
DisplayInterface.cpp	Source code file for display services	127
DisplayInterface.h	DisplayInterface class header file	128
I2C.cpp	Two-wire interface (I2C) source file	131
I2C.h	I2C class header file	132

keepAliveLed.cpp	Definition of function for class <code>keepAliveLed</code>	135
keepAliveLed.h	Class <code>keepAliveLed</code> header file	135
LCD.cpp	<code>LCD</code> class source file	137
LCD.h	<code>LCD</code> class header file	138
LinkedList.cpp		150
LinkedList.h	Linked List library header file	150
main.cpp	Background task file	152
main.h	Background task header file	155
operators.cpp	C++ operators definitions	156
operators.h	C++ operators definitions header file	157
scheduler.cpp	Defines scheduler class	159
scheduler.h	Scheduler class header file	161
TempSensor.cpp	Defines function of class <code>TempSensor</code>	163
TempSensor.h	Class <code>TempSensor</code> header file	164
timer.cpp	Defines function for class timer	166
timer.h	Timer class header file	167
uart.cpp	BSW library for USART	168
uart.h	Header file for USART library	168

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 getCurrentCPULoad \(\)](#)
Get current CPU load value.
- [uint8_t getAverageCPULoad \(\)](#)
Get average CPU load value.
- [uint8_t getMaxCPULoad \(\)](#)
Get maximum CPU load value.

Private Attributes

- [uint8_t current_load](#)
- [uint8_t avg_load](#)
- [uint8_t max_load](#)
- [uint8_t sample_cnt](#)
- [uint8_t sample_mem \[NB_OF_SAMPLES\]](#)
- [uint8_t sample_idx](#)
- [uint16_t last_sum_value](#)

3.1.1 Detailed Description

Class defining CPU load libraries.

This class defines tools to compute and monitor CPU load.

Definition at line 19 of file CpuLoad.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 CpuLoad()

`CpuLoad::CpuLoad ()`

`CpuLoad` class constructor.

This function initializes class `CpuLoad`

Returns

Nothing

Definition at line 13 of file `CpuLoad.cpp`.

3.1.3 Member Function Documentation

3.1.3.1 ComputeCPULoad()

`void CpuLoad::ComputeCPULoad ()`

Computes current CPU load.

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

Returns

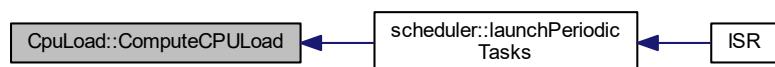
Nothing

Definition at line 27 of file `CpuLoad.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.3.2 getAverageCPULoad()

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

Get average CPU load value.

This function returns the average CPU load value

Returns

Average CPU load value

Definition at line 56 of file CpuLoad.h.

Here is the caller graph for this function:



3.1.3.3 getCurrentCPULoad()

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

Get current CPU load value.

This function returns the current CPU load value

Returns

Current CPU load value

Definition at line 45 of file CpuLoad.h.

Here is the caller graph for this function:



3.1.3.4 getMaxCPUload()

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

Get maximum CPU load value.

This function returns the maximum CPU load value

Returns

Maximum CPU load value

Definition at line 67 of file CpuLoad.h.

Here is the caller graph for this function:



3.1.4 Member Data Documentation

3.1.4.1 avg_load

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

Average CPU load based on the last 50 cycles

Definition at line 74 of file CpuLoad.h.

3.1.4.2 current_load

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

Current CPU load (load of last cycle)

Definition at line 73 of file CpuLoad.h.

3.1.4.3 last_sum_value

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

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

Definition at line 79 of file CpuLoad.h.

3.1.4.4 max_load

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

Maximum CPU load since power on

Definition at line 75 of file CpuLoad.h.

3.1.4.5 sample_cnt

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

Number of samples used to compute average load

Definition at line 76 of file CpuLoad.h.

3.1.4.6 sample_idx

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

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

Definition at line 78 of file CpuLoad.h.

3.1.4.7 sample_mem

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

Memorization of the last NB_OF_SAMPLES measures

Definition at line 77 of file CpuLoad.h.

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

- [CpuLoad.h](#)
- [CpuLoad.cpp](#)

3.2 dht22 Class Reference

DHT 22 driver class.

```
#include <dht22.h>
```

Public Member Functions

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

Private Member Functions

- [void initializeCommunication \(\)](#)
Initializes the communication.

3.2.1 Detailed Description

DHT 22 driver class.

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

Definition at line 22 of file dht22.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 dht22()

```
dht22::dht22 ( )
```

[dht22](#) class constructor

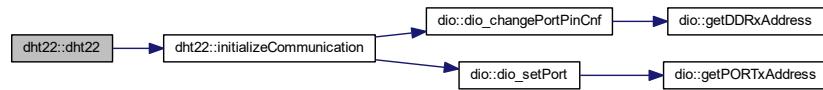
Initializes the class [dht22](#)

Returns

Nothing

Definition at line 22 of file dht22.cpp.

Here is the call graph for this function:



3.2.3 Member Function Documentation

3.2.3.1 initializeCommunication()

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

Initializes the communication.

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

Returns

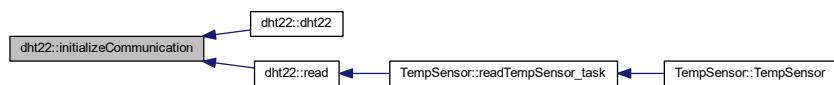
Nothing

Definition at line 198 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.3.2 read()

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

Reads the data from DHT22.

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

Parameters

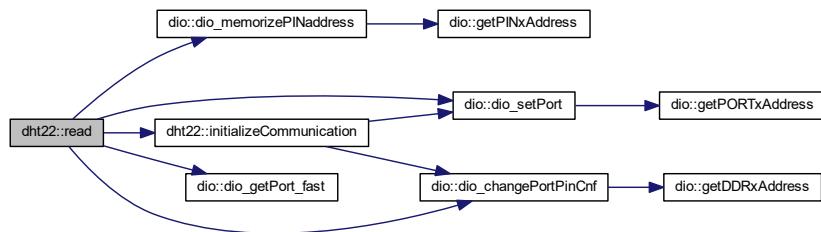
<code>out</code>	<code>raw_humidity</code>	Raw humidity value received from sensor
<code>out</code>	<code>raw_temperature</code>	Raw temperature value received from sensor

Returns

Validity of the read value

Definition at line 27 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



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

- [dht22.h](#)
- [dht22.cpp](#)

3.3 dio Class Reference

DIO class.

```
#include <dio.h>
```

Public Member Functions

- `dio ()`
dio class constructor
- `void dio_setPort (uint8_t portcode, bool state)`
Port setting function.
- `void dio_invertPort (uint8_t portcode)`
Inverts the state of output port.
- `bool dio_getPort (uint8_t portcode)`
Gets the logical state of selected pin.
- `bool dio_getPort_fast (void)`
Gets the logical state of the memorized pin.
- `void dio_changePortPinCnf (uint8_t portcode, uint8_t cnf)`
Changes the IO configuration of the selected pin.
- `void dio_memorizePINaddress (uint8_t portcode)`
Memorizes PINx register address and pin index.

Private Member Functions

- `void ports_init ()`
Digital ports hardware initialization function.
- `uint8_t * getPORTxAddress (uint8_t portcode)`
Gets the physical address of the requested register PORTx.
- `uint8_t * getPINxAddress (uint8_t portcode)`
Gets the physical address of the requested register PINx.
- `uint8_t * getDDRxAddress (uint8_t portcode)`
Gets the physical address of the requested register DDRx.

Private Attributes

- `uint8_t * PINx_addr_mem`
- `uint8_t PINx_idx_mem`

3.3.1 Detailed Description

DIO class.

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

Definition at line 31 of file dio.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 dio()

`dio::dio ()`

dio class constructor

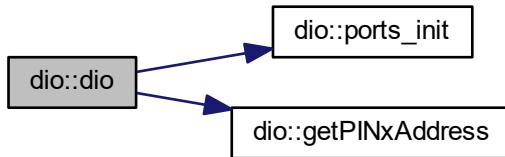
Initializes class dio and calls DIO hardware initialization function

Returns

Nothing

Definition at line 112 of file dio.cpp.

Here is the call graph for this function:



3.3.3 Member Function Documentation

3.3.3.1 dio_changePortPinCnf()

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

Changes the IO configuration of the selected pin.

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

Parameters

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

Returns

Nothing

Definition at line 149 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.3.3.2 dio_getPort()**

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

Gets the logical state of selected pin.

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

Parameters

in	<i>portcode</i>	Encoded pin and register index
----	-----------------	--------------------------------

Returns

Logical state of selected pin

Definition at line 139 of file dio.cpp.

Here is the call graph for this function:



3.3.3.3 dio_getPort_fast()

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

Gets the logical state of the memorized pin.

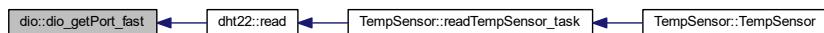
This function gets the logical value of the memorized pin. The corresponding port and pin index are stored in class members PINx_addr_mem and PINx_idx_mem. This mechanism is used to speed up reading time as this function no longer needs to extract register address and pin index from portcode.

Returns

Logical state of selected pin

Definition at line 171 of file dio.cpp.

Here is the caller graph for this function:



3.3.3.4 dio_invertPort()

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

Inverts the state of output port.

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

Parameters

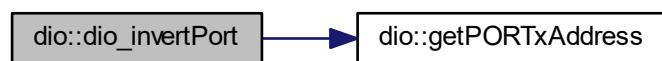
in	<i>portcode</i>	Encoded pin and register index
----	-----------------	--------------------------------

Returns

Nothing

Definition at line 131 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.3.3.5 dio_memorizePINaddress()**

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

Memorizes PINx register address and pin index.

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

Parameters

in	<i>portcode</i>	Encoded pin and register index
----	-----------------	--------------------------------

Returns

Nothing

Definition at line 165 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.3.3.6 dio_setPort()

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

Port setting function.

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

Parameters

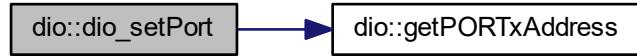
in	<i>portcode</i>	Encoded pin and register index
in	<i>state</i>	Requested state to set pin

Returns

Nothing

Definition at line 121 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.3.3.7 getDDRxAddress()

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

Gets the physical address of the requested register DDRx.

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

Parameters

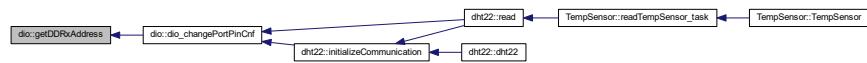
in	<i>portcode</i>	Encoded port code
----	-----------------	-------------------

Returns

Pointer to the DDRx register

Definition at line 83 of file dio.cpp.

Here is the caller graph for this function:



3.3.3.8 getPINxAddress()

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

Gets the physical address of the requested register PINx.

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

Parameters

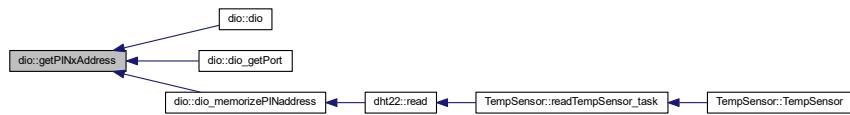
in	<i>portcode</i>	Encoded port code
----	-----------------	-------------------

Returns

Pointer to the PINx register

Definition at line 54 of file dio.cpp.

Here is the caller graph for this function:



3.3.3.9 getPORTxAddress()

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

Gets the physical address of the requested register PORTx.

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

Parameters

in	<i>portcode</i>	Encoded port code
----	-----------------	-------------------

Returns

Pointer to the PORTx register

Definition at line 25 of file dio.cpp.

Here is the caller graph for this function:



3.3.3.10 ports_init()

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

Digital ports hardware initialization function.

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

Returns

Nothing

Definition at line 16 of file dio.cpp.

Here is the caller graph for this function:



3.3.4 Member Data Documentation

3.3.4.1 PINx_addr_mem

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

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

Definition at line 146 of file dio.h.

3.3.4.2 PINx_idx_mem

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

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

Definition at line 147 of file dio.h.

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

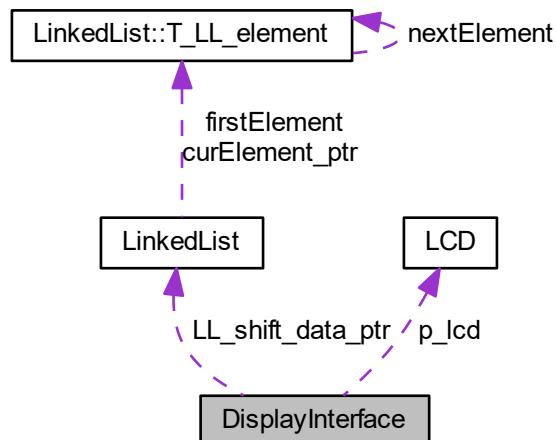
- [dio.h](#)
- [dio.cpp](#)

3.4 DisplayInterface Class Reference

Display interface services class.

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

Collaboration diagram for DisplayInterface:



Public Member Functions

- [DisplayInterface \(\)](#)
Class constructor.
- [bool DisplayFullLine \(uint8_t *str, uint8_t size, uint8_t line, T_DisplayInterface_LineDisplayMode mode\)](#)
Line display function.
- [bool ClearLine \(uint8_t line\)](#)
Line clearing function.
- [bool IsLineEmpty \(uint8_t line\)](#)
Empty line get function.
- [LinkedList * getLLShiftDataPtr \(\)](#)
Linked list shift data get function.

Static Public Member Functions

- static bool [LLElementCompare](#) (void *LLElement, void *CompareElement)
Linked list comparison function.
- static void [shiftLine_task](#) ()
Line shifting periodic task.

Private Member Functions

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

Private Attributes

- LCD * [p_lcd](#)
- uint32_t [dummy](#)
- bool [lineEmptyTab](#) [LCD_SIZE_NB_LINES]
- [LinkedList](#) * [LL_shift_data_ptr](#)

3.4.1 Detailed Description

Display interface services class.

This class defines the services used for interfacing display management services and [LCD](#) screen driver
 Definition at line 68 of file [DisplayInterface.h](#).

3.4.2 Constructor & Destructor Documentation

3.4.2.1 [DisplayInterface\(\)](#)

`DisplayInterface::DisplayInterface ()`

Class constructor.

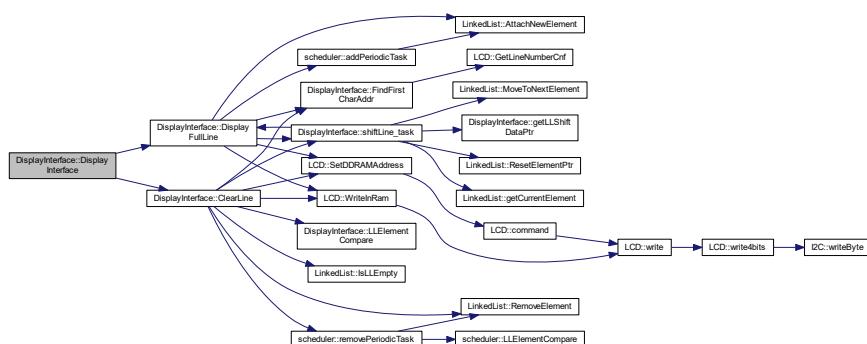
This function initializes all class variables and instantiates the [LCD](#) driver according to the defined configuration.

Returns

Nothing

Definition at line 16 of file [DisplayInterface.cpp](#).

Here is the call graph for this function:



3.4.3 Member Function Documentation

3.4.3.1 ClearLine()

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

Line clearing function.

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

Parameters

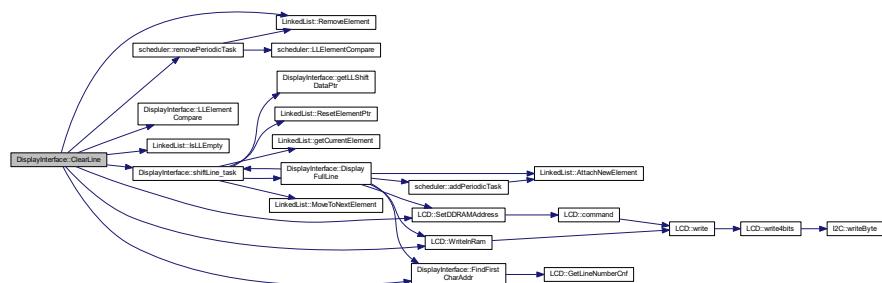
in	<i>line</i>	Line to clear
----	-------------	---------------

Returns

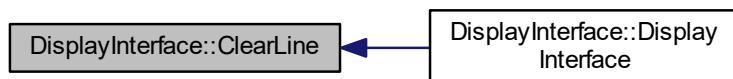
True if the line has been cleared, false otherwise

Definition at line 157 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.2 DisplayFullLine()

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

Line display function.

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

Parameters

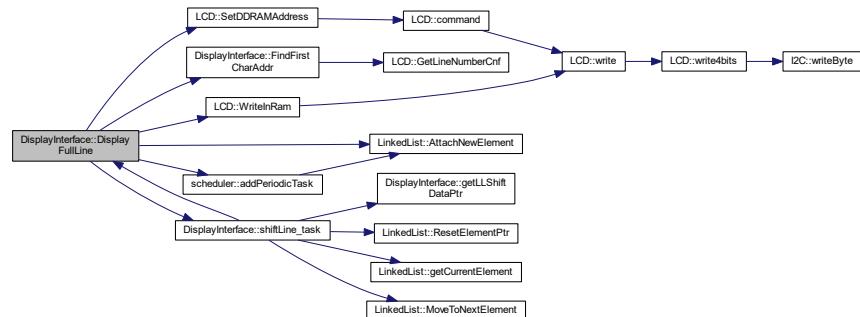
in	<i>str</i>	Pointer to the string to display
in	<i>size</i>	Size of the string to display
in	<i>line</i>	Index of the line where the string shall be displayed
in	<i>mode</i>	Display mode

Returns

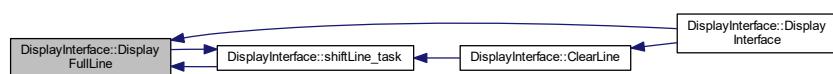
True if the line has been correctly displayed, false otherwise

Definition at line 49 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.3 FindFirstCharAddr()

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

Finds start address of a line.

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

Parameters

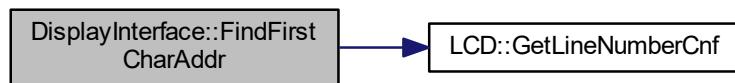
<code>in</code>	<code>line</code>	Line which address shall be found
-----------------	-------------------	-----------------------------------

Returns

Address in DDRAM of the first character of the line

Definition at line 118 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.4 getLLShiftDataPtr()

```
LinkedList* DisplayInterface::getLLShiftDataPtr ( ) [inline]
```

Linked list shift data get function.

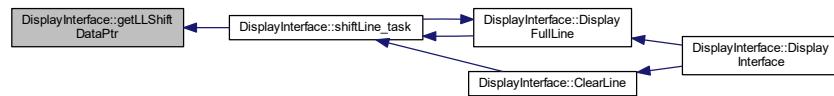
This function returns the pointer to the shift data linked list.

Returns

Pointer to linked list

Definition at line 135 of file `DisplayInterface.h`.

Here is the caller graph for this function:

**3.4.3.5 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

<code>in</code>	<code>line</code>	Requested line
-----------------	-------------------	----------------

Returns

True if the line is empty, false otherwise

Definition at line 188 of file `DisplayInterface.cpp`.

3.4.3.6 LLElementCompare()

```
bool DisplayInterface::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 `DisplayInterface`, the `LLElement` is a shift data pointer (containing a line number inside it), and the `compareElement` a line number. The comparison will be done between the two function pointer.

Parameters

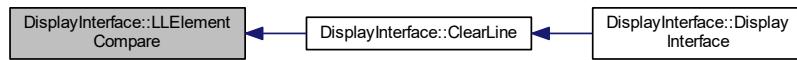
in	<i>LLElement</i>	Pointer to the linked list element
in	<i>CompareElement</i>	Pointer to the element to the compare

Returns

True if both elements are identical, false otherwise

Definition at line 197 of file DisplayInterface.cpp.

Here is the caller graph for this function:



3.4.3.7 shiftLine_task()

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

Line shifting periodic task.

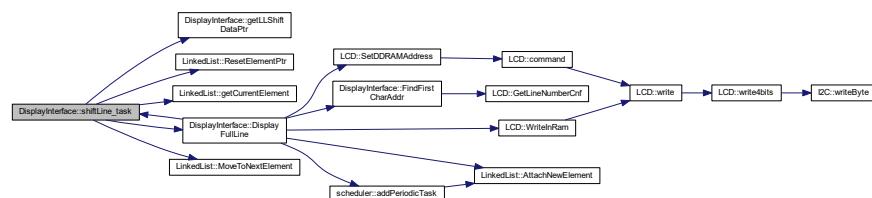
This function is called periodically by the scheduler. It shifts all the lines in line shifting mode and updates the data structures.

Returns

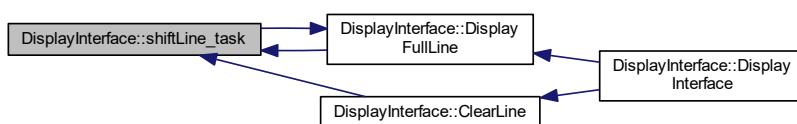
Nothing

Definition at line 208 of file DisplayInterface.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 dummy

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

Needed for data alignment

Definition at line 143 of file DisplayInterface.h.

3.4.4.2 lineEmptyTab

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

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

Definition at line 144 of file DisplayInterface.h.

3.4.4.3 LL_shift_data_ptr

```
LinkedList* DisplayInterface::LL_shift_data_ptr [private]
```

Linked list containing data for line shifting, each element of the list corresponds to a line of the screen

Definition at line 145 of file DisplayInterface.h.

3.4.4.4 p_lcd

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

Pointer to the attached LCD driver object

Definition at line 142 of file DisplayInterface.h.

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

- [DisplayInterface.h](#)
- [DisplayInterface.cpp](#)

3.5 I2C Class Reference

Two-wire serial interface ([I2C](#)) class definition.

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

Public Member Functions

- [I2C](#) (uint32_t *l_bitrate*)

I2C class constructor.
- bool [writeByte](#) (uint8_t **data*)

Byte sending function.
- void [setTxAddress](#) (uint8_t *address*)

Setting function for Tx I2C address.
- void [setBitRate](#) (uint32_t *l_bitrate*)

Variable bitrate setting function.

Private Member Functions

- void [initializeBus](#) ()

I2C bus initialization.

Private Attributes

- uint8_t *tx_address*
- uint32_t *bitrate*

3.5.1 Detailed Description

Two-wire serial interface ([I2C](#)) class definition.

This class manages [I2C](#) driver.

Definition at line 23 of file I2C.h.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 [I2C\(\)](#)

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

[I2C](#) class constructor.

This function initializes the [I2C](#) class and calls bus initialization function

Parameters

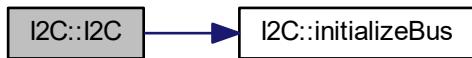
in	<i>I_bitrate</i>	Requested bitrate for I2C bus (in Hz)
----	------------------	---------------------------------------

Returns

Nothing

Definition at line 15 of file I2C.cpp.

Here is the call graph for this function:



3.5.3 Member Function Documentation

3.5.3.1 initializeBus()

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

I2C bus initialization.

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

Returns

Nothing

Definition at line 76 of file I2C.cpp.

Here is the caller graph for this function:



3.5.3.2 setBitRate()

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

Variable bitrate setting function.

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

Parameters

in	<i>l_bitrate</i>	Requested bitrate (in Hz)
----	------------------	---------------------------

Returns

Nothing

Definition at line 71 of file I2C.cpp.

3.5.3.3 setTxAddress()

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

Setting function for Tx **I2C** address.

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

Parameters

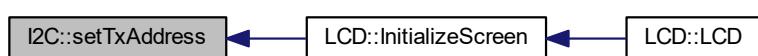
in	<i>address</i>	Requested Tx address
----	----------------	----------------------

Returns

Nothing

Definition at line 66 of file I2C.cpp.

Here is the caller graph for this function:



3.5.3.4 writeByte()

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

Byte sending function.

This function sends one byte on [I2C](#) bus

Parameters

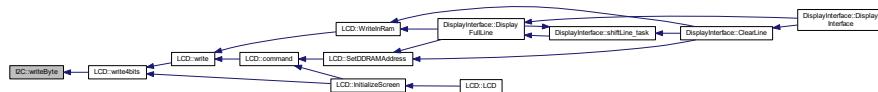
<code>in</code>	<code>data</code>	Pointer to the data to send
-----------------	-------------------	-----------------------------

Returns

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

Definition at line 23 of file I2C.cpp.

Here is the caller graph for this function:



3.5.4 Member Data Documentation

3.5.4.1 bitrate

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

Definition at line 63 of file I2C.h.

3.5.4.2 tx_address

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

Definition at line 62 of file I2C.h.

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

- [I2C.h](#)
- [I2C.cpp](#)

3.6 keepAliveLed Class Reference

Class for keep-alive LED blinking.

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

Public Member Functions

- [keepAliveLed \(\)](#)

Class constructor.

Static Public Member Functions

- static void [blinkLed_task \(\)](#)

Task for LED blinking.

3.6.1 Detailed Description

Class for keep-alive LED blinking.

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

Definition at line 22 of file keepAliveLed.h.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 [keepAliveLed\(\)](#)

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

Class constructor.

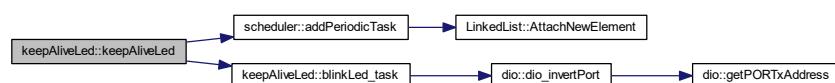
This function initializes the class keepAliveLed

Returns

Nothing

Definition at line 15 of file keepAliveLed.cpp.

Here is the call graph for this function:



3.6.3 Member Function Documentation

3.6.3.1 blinkLed_task()

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

Task for LED blinking.

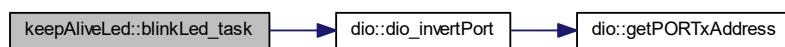
This function is inserted into the scheduler. It changes the state of the LED output to make it blink

Returns

Nothing

Definition at line 21 of file `keepAliveLed.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



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

- [keepAliveLed.h](#)
- [keepAliveLed.cpp](#)

3.7 LCD Class Reference

Class for [LCD](#) S2004A display driver.

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

Public Member Functions

- `LCD (const T_LCD_conf_struct *init_conf)`
`LCD class constructor.`
- `void command (T_LCD_command cmd)`
`LCD command management function.`
- `void ConfigureBacklight (bool enable)`
`Backlight configuration function.`
- `void ConfigureLineNumber (bool param)`
`Line type configuration function.`
- `void ConfigureFontType (bool param)`
`Font configuration function.`
- `void ConfigureDisplayOnOff (bool param)`
`Display configuration function.`
- `void ConfigureCursorOnOff (bool param)`
`Cursor configuration function.`
- `void ConfigureCursorBlink (bool param)`
`Cursor blinking configuration function.`
- `void ConfigureEntryModeDir (bool param)`
`Entry mode direction configuration function.`
- `void ConfigureEntryModeShift (bool param)`
`Entry mode shift configuration function.`
- `void 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`
- `uint8_t ddrum_addr`

3.7.1 Detailed Description

Class for [LCD](#) S2004A display driver.

This class handles functions managing [LCD](#) display S2004a on [I2C](#) bus

Definition at line 143 of file LCD.h.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 LCD()

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

[LCD](#) class constructor.

This constructor function initializes the class [LCD](#) and calls screen configuration function.

Parameters

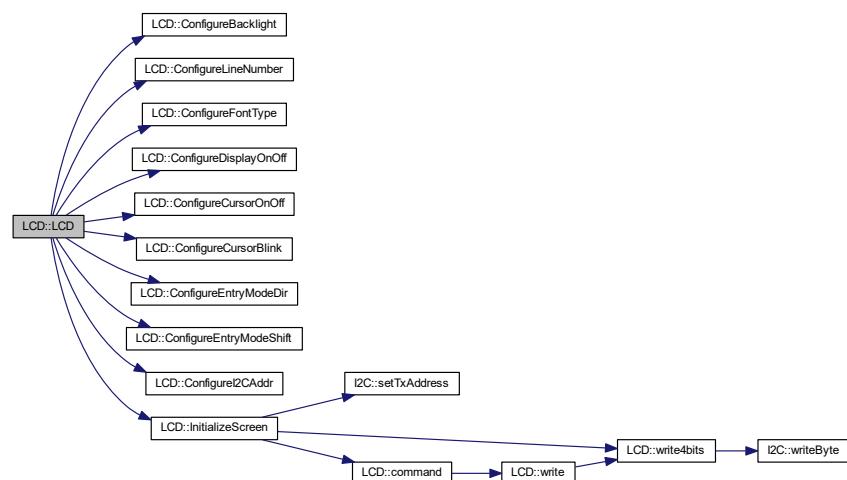
in	<i>init_conf</i>	Initial configuration structure
----	------------------	---------------------------------

Returns

Nothing

Definition at line 14 of file LCD.cpp.

Here is the call graph for this function:



3.7.3 Member Function Documentation

3.7.3.1 command()

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

LCD command management function.

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

Parameters

in	<i>cmd</i>	Requested command
----	------------	-------------------

Returns

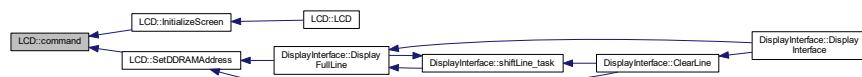
Nothing

Definition at line 115 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.3.2 ConfigureBacklight()

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

Backlight configuration function.

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

Parameters

in	<i>enable</i>	True if backlight shall be on, False otherwise
----	---------------	--

Returns

Nothing

Definition at line 173 of file LCD.h.

Here is the caller graph for this function:



3.7.3.3 ConfigureCursorBlink()

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

Cursor blinking configuration function.

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

Parameters

in	<i>param</i>	Configuration value
----	--------------	---------------------

Returns

Nothing

Definition at line 233 of file LCD.h.

Here is the caller graph for this function:



3.7.3.4 ConfigureCursorOnOff()

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

Cursor configuration function.

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

Parameters

in	param	Configuration value
----	-------	---------------------

Returns

Nothing

Definition at line 221 of file LCD.h.

Here is the caller graph for this function:



3.7.3.5 ConfigureDisplayOnOff()

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

Display configuration function.

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

Parameters

in	param	Configuration value
----	-------	---------------------

Returns

Nothing

Definition at line 209 of file LCD.h.

Here is the caller graph for this function:



3.7.3.6 ConfigureEntryModeDir()

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

Entry mode direction configuration function.

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

Parameters

in	param	Configuration value
----	-------	---------------------

Returns

Nothing

Definition at line 245 of file LCD.h.

Here is the caller graph for this function:



3.7.3.7 ConfigureEntryModeShift()

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

Entry mode shift configuration function.

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

Parameters

in	<i>param</i>	Configuration value
----	--------------	---------------------

Returns

Nothing

Definition at line 257 of file LCD.h.

Here is the caller graph for this function:



3.7.3.8 ConfigureFontType()

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

Font configuration function.

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

Parameters

in	<i>param</i>	Configuration value
----	--------------	---------------------

Returns

Nothing

Definition at line 197 of file LCD.h.

Here is the caller graph for this function:



3.7.3.9 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	<i>param</i>	I2C address
----	--------------	-------------

Returns

Nothing

Definition at line 269 of file LCD.h.

Here is the caller graph for this function:



3.7.3.10 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

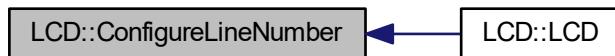
in	param	Configuration value
----	-------	---------------------

Returns

Nothing

Definition at line 185 of file LCD.h.

Here is the caller graph for this function:



3.7.3.11 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 289 of file LCD.h.

3.7.3.12 GetLineNumberCnf()

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

Number of line get function.

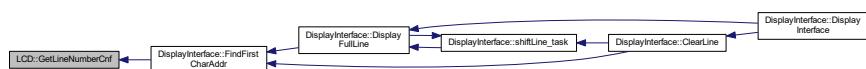
This function returns the line number configuration of the screen : 1 or 2 lines mode.

Returns

Line number configuration

Definition at line 311 of file LCD.h.

Here is the caller graph for this function:



3.7.3.13 InitializeScreen()

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

Screen configuration function.

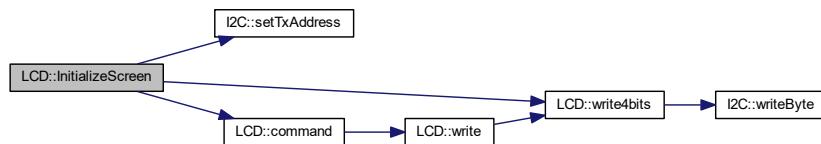
This function configures the [LCD](#) screen. It's must be called during initialization phase, or the screen won't be usable. The configuration process is described in [LCD](#) datasheet J2004A-GFDN-DYNC

Returns

Nothing

Definition at line 63 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.3.14 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	<i>addr</i>	New DDRAM address
----	-------------	-------------------

Returns

Nothing

Definition at line 158 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.3.15 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	<i>data</i>	8-bit data for D0-7 pins of screen
in	<i>mode</i>	Requested mode for LCD communication

Returns

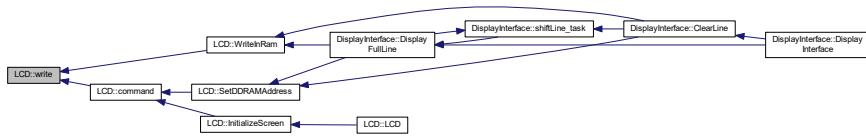
Nothing

Definition at line 52 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.3.16 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

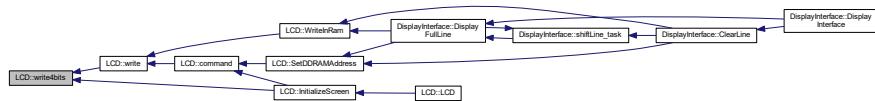
Nothing

Definition at line 35 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.3.17 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	<i>a_char</i>	Data byte to write in RAM
in	<i>area</i>	Area in RAM where the data will be written : DDRAM or CGRAM

Returns

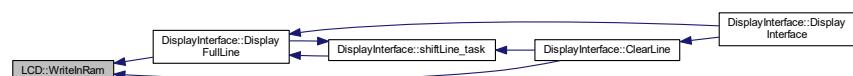
Nothing

Definition at line 180 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.4 Member Data Documentation

3.7.4.1 `backlight_enable`

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

Backlight enable flag

Definition at line 319 of file LCD.h.

3.7.4.2 `cnfCursorBlink`

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

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

Definition at line 324 of file LCD.h.

3.7.4.3 `cnfCursorOnOff`

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

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

Definition at line 323 of file LCD.h.

3.7.4.4 `cnfDisplayOnOff`

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

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

Definition at line 322 of file LCD.h.

3.7.4.5 `cnfEntryModeDir`

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

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

Definition at line 325 of file LCD.h.

3.7.4.6 cnfEntryModeShift

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

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

Definition at line 326 of file LCD.h.

3.7.4.7 cnfFontType

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

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

Definition at line 321 of file LCD.h.

3.7.4.8 cnfI2C_addr

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

I2C address of the [LCD](#) screen

Definition at line 327 of file LCD.h.

3.7.4.9 cnfLineNumber

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

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

Definition at line 320 of file LCD.h.

3.7.4.10 ddram_addr

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

Screen DDRAM address

Definition at line 329 of file LCD.h.

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

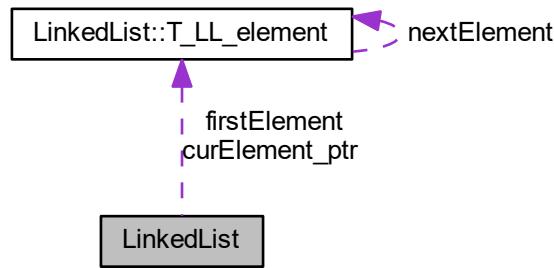
- [LCD.h](#)
- [LCD.cpp](#)

3.8 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.
- 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.

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`

3.8.1 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`.

3.8.2 Member Typedef Documentation

3.8.2.1 `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.

3.8.3 Constructor & Destructor Documentation

3.8.3.1 `LinkedList()`

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

Class constructor.

This constructor initializes a linked list

Returns

Nothing

Definition at line 12 of file `LinkedList.cpp`.

3.8.4 Member Function Documentation

3.8.4.1 `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

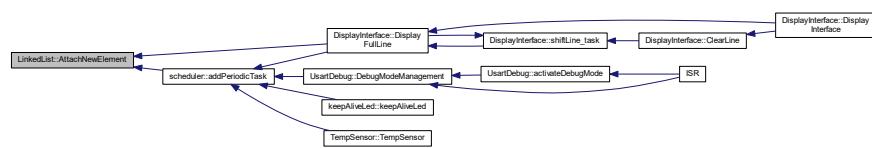
in	<i>data_ptr</i>	Pointer to the data element
----	-----------------	-----------------------------

Returns

Nothing

Definition at line 18 of file `LinkedList.cpp`.

Here is the caller graph for this function:

**3.8.4.2 getCurrentElement()**

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

Current element get function.

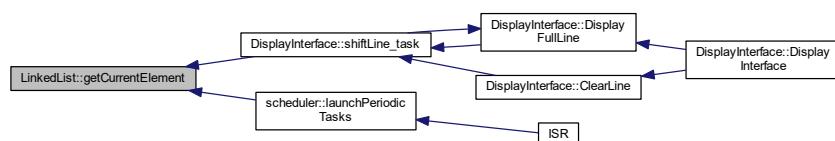
This function returns a pointer to the current pointed data in the chain.

Returns

Pointer to the current data

Definition at line 59 of file `LinkedList.h`.

Here is the caller graph for this function:



3.8.4.3 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 84 of file LinkedList.cpp.

Here is the caller graph for this function:



3.8.4.4 MoveToNextElement()

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

Move to next element function.

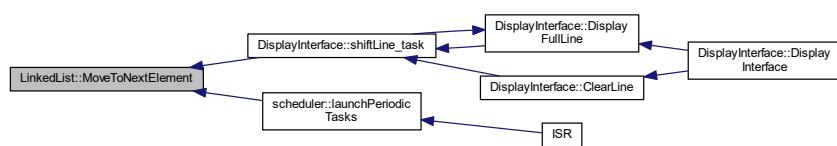
This function moves the element pointer to the next element of the chain.

Returns

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

Definition at line 70 of file LinkedList.cpp.

Here is the caller graph for this function:



3.8.4.5 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

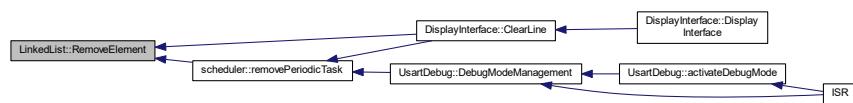
in	<i>comparisonFct_ptr</i>	Pointer to the comparison function to use
in	<i>reference_ptr</i>	Pointer to the reference data used for comparison

Returns

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

Definition at line 43 of file LinkedList.cpp.

Here is the caller graph for this function:

**3.8.4.6 ResetElementPtr()**

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

Resets element pointer.

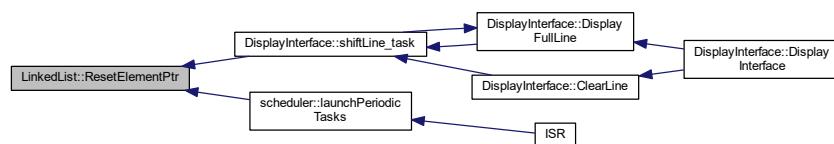
This function sets the element pointer to the first element of the chain.

Returns

Nothing

Definition at line 78 of file LinkedList.h.

Here is the caller graph for this function:

**3.8.5 Member Data Documentation**

3.8.5.1 curElement_ptr

`T_LL_element* LinkedList::curElement_ptr [private]`

Definition at line 105 of file `LinkedList.h`.

3.8.5.2 firstElement

`T_LL_element* LinkedList::firstElement [private]`

Definition at line 104 of file `LinkedList.h`.

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

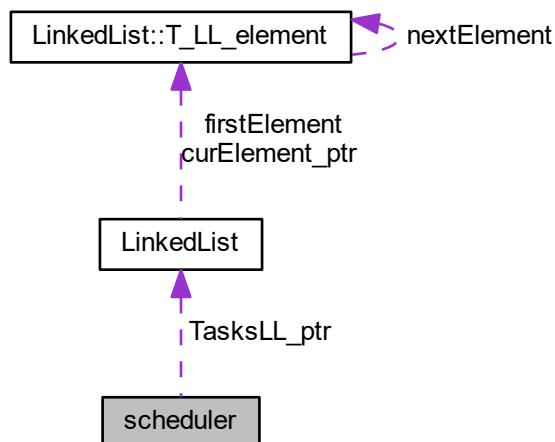
- [LinkedList.h](#)
- [LinkedList.cpp](#)

3.9 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.

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

- `LinkedList * TasksLL_ptr`
- `uint32_t pit_number`

3.9.1 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 31 of file scheduler.h.

3.9.2 Member Typedef Documentation

3.9.2.1 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.

3.9.3 Constructor & Destructor Documentation

3.9.3.1 scheduler()

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

scheduler class constructor

This function initializes the class scheduler

Returns

Nothing

Definition at line 20 of file scheduler.cpp.

Here is the call graph for this function:



3.9.4 Member Function Documentation

3.9.4.1 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	<i>task_ptr</i>	Pointer to the task which will be added
in	<i>a_period</i>	Period of the new task

Returns

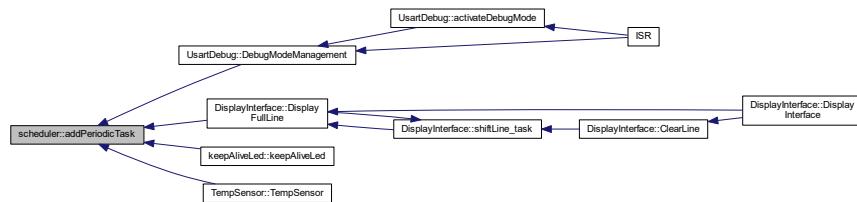
Nothing

Definition at line 70 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.9.4.2 getPitNumber()**

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

Get function for PIT number.

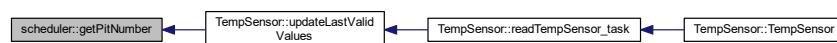
This function returns the PIT number

Returns

PIT number

Definition at line 82 of file scheduler.cpp.

Here is the caller graph for this function:



3.9.4.3 launchPeriodicTasks()

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

Main scheduler function.

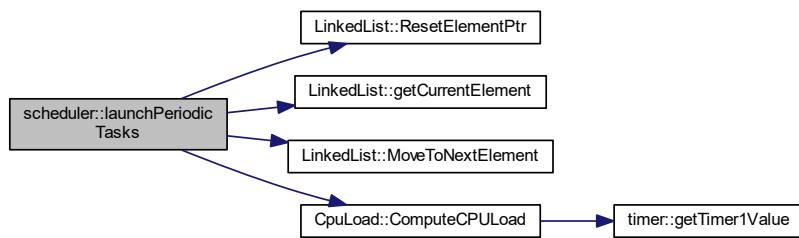
This function launches the scheduled tasks according to current software time and task configuration

Returns

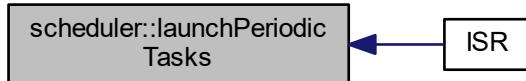
Nothing

Definition at line 32 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.9.4.4 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

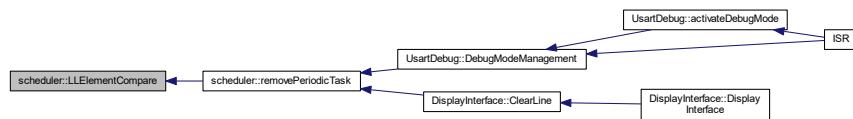
in	<i>LLElement</i>	Pointer to the linked list element
in	<i>CompareElement</i>	Pointer to the element to the compare

Returns

True if both elements are identical, false otherwise

Definition at line 93 of file scheduler.cpp.

Here is the caller graph for this function:

**3.9.4.5 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

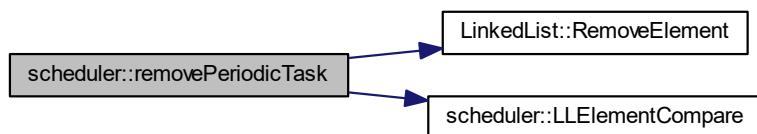
in	<i>task_ptr</i>	address of the task to remove from scheduler
----	-----------------	--

Returns

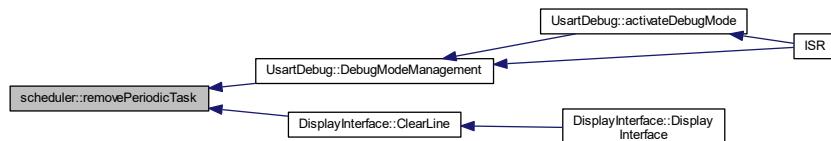
TRUE if the task has been removed, FALSE if the task does not exist in the scheduler

Definition at line 88 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.9.4.6 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 64 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.9.5 Member Data Documentation

3.9.5.1 pit_number

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

Counter of periodic interrupts

Definition at line 117 of file scheduler.h.

3.9.5.2 TasksLL_ptr

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

Pointer to the linked list object containing the tasks

Definition at line 115 of file scheduler.h.

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

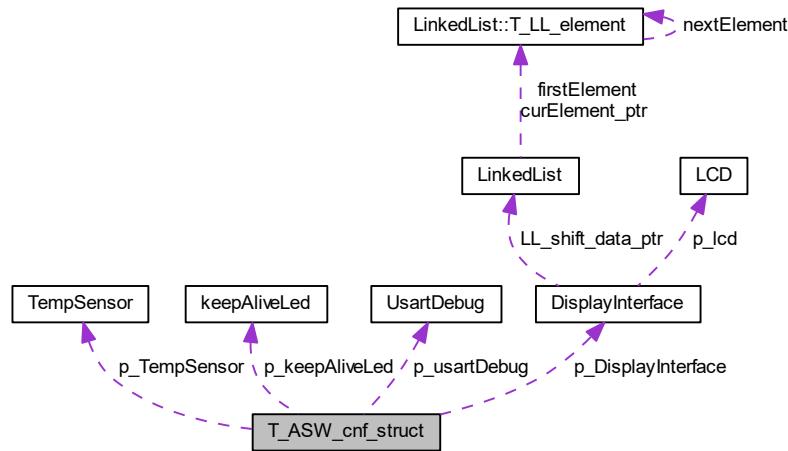
- [scheduler.h](#)
- [scheduler.cpp](#)

3.10 T_ASW_cnf_struct Struct Reference

ASW configuration structure.

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

Collaboration diagram for T_ASW_cnf_struct:



Public Attributes

- `UsartDebug * p_usartDebug`
- `keepAliveLed * p_keepAliveLed`
- `TempSensor * p(TempSensor)`
- `DisplayInterface * p_DisplayInterface`

3.10.1 Detailed Description

ASW configuration structure.

This structure contains all pointers to instanced applicative objects

Definition at line 24 of file `asw.h`.

3.10.2 Member Data Documentation

3.10.2.1 p_DisplayInterface

`DisplayInterface* T_ASW_cnf_struct::p_DisplayInterface`

Pointer to [DisplayInterface](#) object

Definition at line 29 of file asw.h.

3.10.2.2 p_keepAliveLed

`keepAliveLed* T_ASW_cnf_struct::p_keepAliveLed`

Pointer to [keepAliveLed](#) object

Definition at line 27 of file asw.h.

3.10.2.3 p_TempSensor

`TempSensor* T_ASW_cnf_struct::p_TempSensor`

Pointer to [TempSensor](#) object

Definition at line 28 of file asw.h.

3.10.2.4 p_usartDebug

`UsartDebug* T_ASW_cnf_struct::p_usartDebug`

Pointer to usart debug object

Definition at line 26 of file asw.h.

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

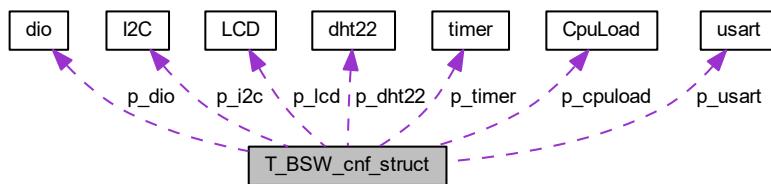
- [asw.h](#)

3.11 T_BSW_cnf_struct Struct Reference

BSW configuration structure.

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

Collaboration diagram for T_BSW_cnf_struct:



Public Attributes

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

3.11.1 Detailed Description

BSW configuration structure.

This structure contains all pointers to instanced drivers objects

Definition at line 33 of file bsw.h.

3.11.2 Member Data Documentation

3.11.2.1 p_cpuload

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

Pointer to cpu load library object

Definition at line 39 of file bsw.h.

3.11.2.2 p_dht22

`dht22* T_BSW_cnf_struct::p_dht22`

Pointer to [dht22](#) driver object

Definition at line 38 of file bsw.h.

3.11.2.3 p_dio

`dio* T_BSW_cnf_struct::p_dio`

Pointer to dio driver object

Definition at line 36 of file bsw.h.

3.11.2.4 p_i2c

`I2C* T_BSW_cnf_struct::p_i2c`

Pointer to [I2C](#) driver object

Definition at line 40 of file bsw.h.

3.11.2.5 p_lcd

`LCD* T_BSW_cnf_struct::p_lcd`

Pointer to [LCD](#) driver object

Definition at line 41 of file bsw.h.

3.11.2.6 p_timer

`timer* T_BSW_cnf_struct::p_timer`

Pointer to timer driver object

Definition at line 37 of file bsw.h.

3.11.2.7 p_usart

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

Pointer to usart driver object

Definition at line 35 of file bsw.h.

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

- [bsw.h](#)

3.12 T_Display_shift_data Struct Reference

Structure containing shift data.

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

Public Attributes

- `uint8_t * str_start_ptr`
- `uint8_t * str_cur_ptr`
- `uint8_t size`
- `uint8_t line`
- `uint8_t temporization`

3.12.1 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 50 of file DisplayInterface.h.

3.12.2 Member Data Documentation

3.12.2.1 line

```
uint8_t T_Display_shift_data::line
```

Definition at line 55 of file DisplayInterface.h.

3.12.2.2 size

```
uint8_t T_Display_shift_data::size
```

Definition at line 54 of file DisplayInterface.h.

3.12.2.3 str_cur_ptr

```
uint8_t* T_Display_shift_data::str_cur_ptr
```

Definition at line 53 of file DisplayInterface.h.

3.12.2.4 str_start_ptr

```
uint8_t* T_Display_shift_data::str_start_ptr
```

Definition at line 52 of file DisplayInterface.h.

3.12.2.5 temporization

```
uint8_t T_Display_shift_data::temporization
```

Definition at line 56 of file DisplayInterface.h.

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

- [DisplayInterface.h](#)

3.13 T_LCD_conf_struct Struct Reference

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

Public Attributes

- 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](#)

3.13.1 Detailed Description

Definition at line 125 of file LCD.h.

3.13.2 Member Data Documentation

3.13.2.1 `backlight_en`

```
bool T_LCD_conf_struct::backlight_en
```

Definition at line 128 of file LCD.h.

3.13.2.2 `cursor_en`

```
bool T_LCD_conf_struct::cursor_en
```

Definition at line 132 of file LCD.h.

3.13.2.3 `cursorBlink_en`

```
bool T_LCD_conf_struct::cursorBlink_en
```

Definition at line 133 of file LCD.h.

3.13.2.4 `display_en`

```
bool T_LCD_conf_struct::display_en
```

Definition at line 131 of file LCD.h.

3.13.2.5 `entryModeDir`

```
bool T_LCD_conf_struct::entryModeDir
```

Definition at line 134 of file LCD.h.

3.13.2.6 entryModeShift

```
bool T_LCD_conf_struct::entryModeShift
```

Definition at line 135 of file LCD.h.

3.13.2.7 fontType_cnf

```
bool T_LCD_conf_struct::fontType_cnf
```

Definition at line 130 of file LCD.h.

3.13.2.8 i2c_addr

```
uint8_t T_LCD_conf_struct::i2c_addr
```

Definition at line 127 of file LCD.h.

3.13.2.9 lineNumber_cnf

```
bool T_LCD_conf_struct::lineNumber_cnf
```

Definition at line 129 of file LCD.h.

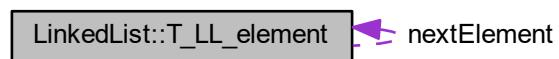
The documentation for this struct was generated from the following file:

- [LCD.h](#)

3.14 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`

3.14.1 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 97 of file `LinkedList.h`.

3.14.2 Member Data Documentation

3.14.2.1 `data_ptr`

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

Definition at line 99 of file `LinkedList.h`.

3.14.2.2 `nextElement`

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

Definition at line 100 of file `LinkedList.h`.

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

- [LinkedList.h](#)

3.15 `scheduler::Task_t` Struct Reference

Type defining a task structure.

Public Attributes

- `TaskPtr_t TaskPtr`
- `uint16_t period`

3.15.1 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 108 of file scheduler.h.

3.15.2 Member Data Documentation

3.15.2.1 period

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

Period of the task

Definition at line 111 of file scheduler.h.

3.15.2.2 TaskPtr

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

Pointer to the task

Definition at line 110 of file scheduler.h.

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

- [scheduler.h](#)

3.16 TempSensor Class Reference

Class for temperature sensor.

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

Public Member Functions

- [TempSensor \(\)](#)
Class constructor.
- [uint16_t * getTempPtr \(\)](#)
Get pointer to data raw_temperature.
- [uint16_t * getHumPtr \(\)](#)
Get pointer to data raw_humidity.
- [bool getTemp \(uint16_t *temp\)](#)
Get temperature data.
- [bool getHumidity \(uint16_t *hum\)](#)
Get humidity data.
- [void setValidity \(bool validity\)](#)
Set data val_validity.
- [void updateLastValidValues \(\)](#)

Static Public Member Functions

- [static void readTempSensor_task \(\)](#)
Task for reading temperature and humidity values.

Private Attributes

- [uint16_t read_temperature](#)
- [uint16_t read_humidity](#)
- [bool validity_last_read](#)
- [bool validity](#)
- [uint32_t valid_pit](#)
- [uint16_t valid_temp](#)
- [uint16_t valid_hum](#)

3.16.1 Detailed Description

Class for temperature sensor.

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

Definition at line 19 of file TempSensor.h.

3.16.2 Constructor & Destructor Documentation

3.16.2.1 TempSensor()

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

Class constructor.

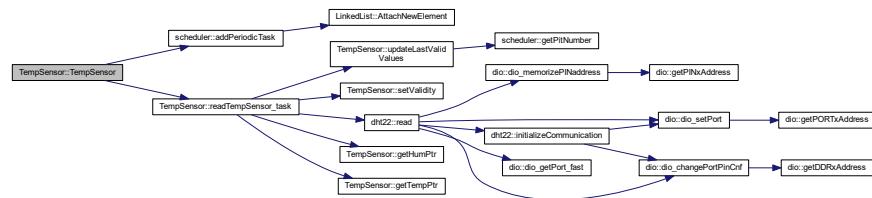
This function initializes all data of the class [TempSensor](#)

Returns

Nothing

Definition at line 16 of file [TempSensor.cpp](#).

Here is the call graph for this function:



3.16.3 Member Function Documentation

3.16.3.1 getHumidity()

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

Get humidity data.

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

Parameters

out	<i>hum</i>	Humidity value
-----	------------	----------------

Returns

Validity of humidity

Definition at line 66 of file [TempSensor.cpp](#).

Here is the caller graph for this function:



3.16.3.2 getHumPtr()

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

Get pointer to data raw_humidity.

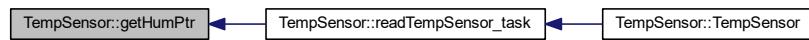
This function returns a pointer to the class member raw_humidity

Returns

Pointer to raw_humidity

Definition at line 41 of file TempSensor.cpp.

Here is the caller graph for this function:



3.16.3.3 getTemp()

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

Get temperature data.

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

Parameters

out	<i>temp</i>	Temperature value
-----	-------------	-------------------

Returns

Validity of temperature

Definition at line 72 of file TempSensor.cpp.

Here is the caller graph for this function:

**3.16.3.4 getTempPtr()**

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

Get pointer to data raw_temperature.

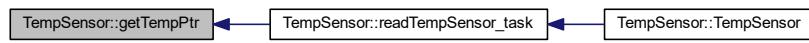
This function returns a pointer to the class member raw_temperature

Returns

Pointer to raw_temperature

Definition at line 46 of file TempSensor.cpp.

Here is the caller graph for this function:

**3.16.3.5 readTempSensor_task()**

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

Task for reading temperature and humidity values.

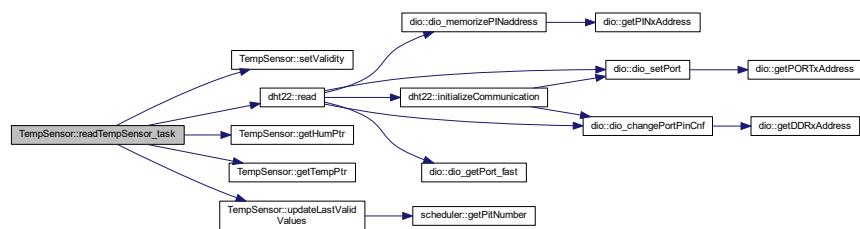
This task reads temperature and humidity data using DHT22 driver. It is called every 5 seconds.

Returns

Nothing

Definition at line 30 of file TempSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.16.3.6 setValidity()

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

Set data val_validity.

This function sets the class member val_validity

Parameters

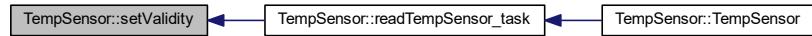
in	<i>validity</i>	Value of validity
----	-----------------	-------------------

Returns

Nothing

Definition at line 36 of file TempSensor.cpp.

Here is the caller graph for this function:

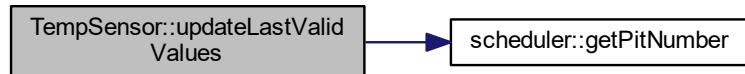


3.16.3.7 updateLastValidValues()

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

Definition at line 51 of file TempSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.16.4 Member Data Documentation

3.16.4.1 read_humidity

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

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

Definition at line 86 of file TempSensor.h.

3.16.4.2 `read_temperature`

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

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

Definition at line 85 of file TempSensor.h.

3.16.4.3 `valid_hum`

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

Valid value of humidity

Definition at line 93 of file TempSensor.h.

3.16.4.4 `valid_pit`

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

pit number of the last time when data were valid

Definition at line 90 of file TempSensor.h.

3.16.4.5 `valid_temp`

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

Valid value of temperature

Definition at line 92 of file TempSensor.h.

3.16.4.6 `validity`

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

validity of official temperature and humidity data

Definition at line 89 of file TempSensor.h.

3.16.4.7 validity_last_read

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

Validity of last read temperature and humidity data

Definition at line 87 of file TempSensor.h.

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

- [TempSensor.h](#)
- [TempSensor.cpp](#)

3.17 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](#)

3.17.1 Detailed Description

Class defining a timer.

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

Definition at line 22 of file timer.h.

3.17.2 Constructor & Destructor Documentation

3.17.2.1 timer()

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

Class constructor.

This function initializes class attributes

Returns

Nothing

Definition at line 13 of file timer.cpp.

3.17.3 Member Function Documentation

3.17.3.1 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	<i>a_prescaler</i>	prescaler value
in	<i>a_ctcValue</i>	Value to which the counter will compare before raising an interrupt

Returns

Nothing

Definition at line 18 of file timer.cpp.

Here is the caller graph for this function:



3.17.3.2 getTimer1Value()

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

Reads current value of timer #1.

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

Returns

Current timer value

Definition at line 61 of file timer.h.

Here is the caller graph for this function:



3.17.3.3 startTimer1()

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

Start Timer #1.

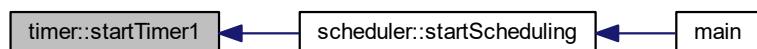
This functions starts Timer #1. Timer shall be initialized before this function is called.

Returns

Nothing

Definition at line 56 of file timer.cpp.

Here is the caller graph for this function:



3.17.3.4 stopTimer1()

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

Stops Timer #1.

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

Returns

Nothing

Definition at line 67 of file timer.cpp.

3.17.4 Member Data Documentation

3.17.4.1 prescaler

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

Definition at line 67 of file timer.h.

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

- [timer.h](#)
- [timer.cpp](#)

3.18 usart Class Reference

USART serial bus class.

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

Public Member Functions

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

Static Private Member Functions

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

Private Attributes

- uint16_t [BaudRate](#)

3.18.1 Detailed Description

USART serial bus class.

This class defines all useful functions for USART serial bus

Definition at line 16 of file usart.h.

3.18.2 Constructor & Destructor Documentation

3.18.2.1 usart()

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

Class usart constructor.

Initializes the class and call hardware initialization function

Parameters

in	<i>a_BaudRate</i>	Desired Baud Rate (16 bit) - up to 57600
----	-------------------	--

Returns

Nothing.

Definition at line 14 of file usart.cpp.

Here is the call graph for this function:



3.18.3 Member Function Documentation

3.18.3.1 setBaudRate()

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

Setting baud rate.

This function sets the attribute BaudRate of the class usart

Parameters

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

Returns

Nothing

Definition at line 63 of file usart.cpp.

3.18.3.2 usart_init()

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

USART hardware initialization.

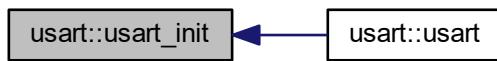
This function will initialize the USART using selected baudrate. User must pay attention to select one of the usually used Baud Rate (9600, 19200, 38400, 57600). Note that since an uint16 is used as argument, Baud rate cannot be more than 57600.

Returns

Nothing.

Definition at line 21 of file usart.cpp.

Here is the caller graph for this function:



3.18.3.3 usart_read()

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

USART read function.

This function will read reception register of USART

Returns

The function returns the 8 bits read from reception buffer

Definition at line 79 of file usart.cpp.

Here is the caller graph for this function:



3.18.3.4 usart_sendString()

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

Sending a string on USART link.

Just write data to the Serial link using usart_transmit function

Parameters

<code>in</code>	<code>str</code>	Pointer to the string being sent
-----------------	------------------	----------------------------------

Returns

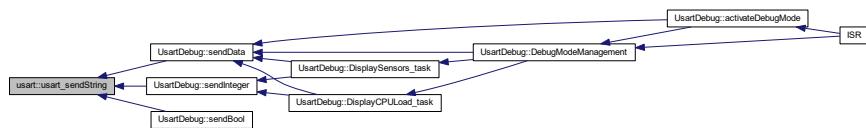
Nothing.

Definition at line 44 of file `uart.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.18.3.5 usart_transmit()**

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

USART Transmit data.

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

Parameters

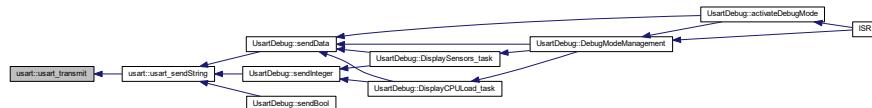
<code>in</code>	<code>Data</code>	Desired data char to transmit
-----------------	-------------------	-------------------------------

Returns

Nothing.

Definition at line 70 of file usart.cpp.

Here is the caller graph for this function:



3.18.4 Member Data Documentation

3.18.4.1 BaudRate

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

Defines the baud rate used by driver

Definition at line 70 of file usart.h.

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

- [usart.h](#)
- [usart.cpp](#)

3.19 UsartDebug Class Reference

Class used for debugging on usart link.

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

Public Member Functions

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

Static Public Member Functions

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

Private Member Functions

- void [sendData \(char *str\)](#)
Send a string on USART link.

Private Attributes

- [debug_state_t debug_state](#)
- bool [debugModeActive_F](#)

3.19.1 Detailed Description

Class used for debugging on usart link.

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

Definition at line 33 of file debug.h.

3.19.2 Constructor & Destructor Documentation

3.19.2.1 UsartDebug()

`UsartDebug::UsartDebug ()`

Class [UsartDebug](#) constructor.

Initializes the class [UsartDebug](#)

Returns

Nothing

Definition at line 31 of file debug.cpp.

3.19.3 Member Function Documentation

3.19.3.1 activateDebugMode()

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

Activates debug mode.

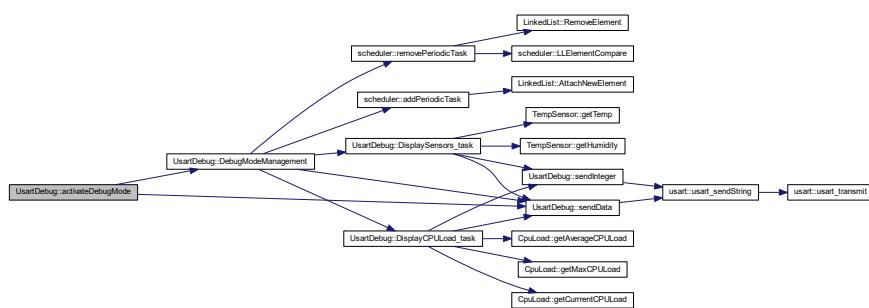
This function activates USART debug mode.

Returns

Nothing

Definition at line 126 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.19.3.2 DebugModeManagement()

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

Management of debug mode.

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

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

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

Parameters

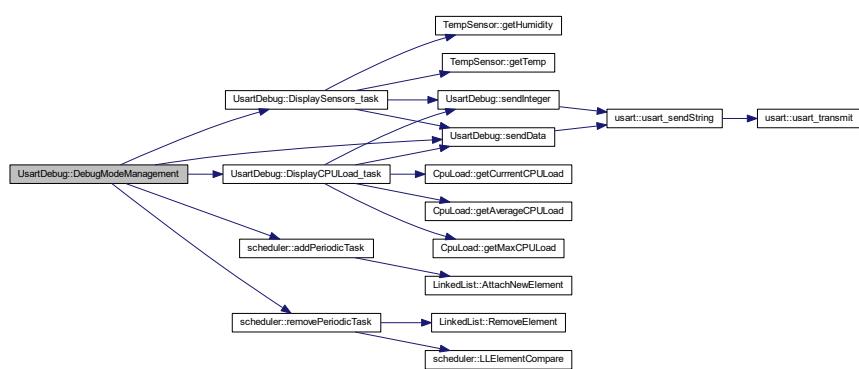
in	<i>rcv_char</i>	8 bits character received on USART
----	-----------------	------------------------------------

Returns

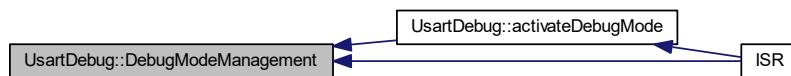
Nothing

Definition at line 134 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.19.3.3 DisplayCPULoad_task()**

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

Displays CPU load data on usart link.

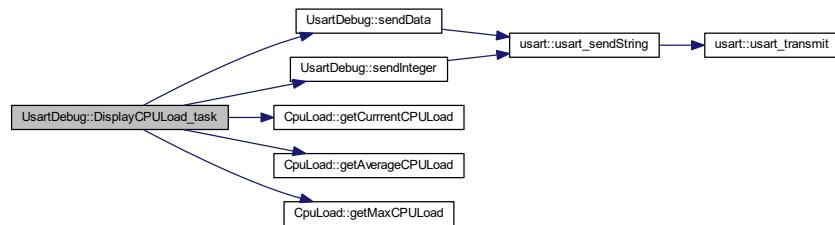
This task sends CPU load data (current and average load) on usart link every 5 seconds

Returns

Nothing

Definition at line 110 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.19.3.4 DisplaySensors_task()**

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

Displays sensors data on usart link.

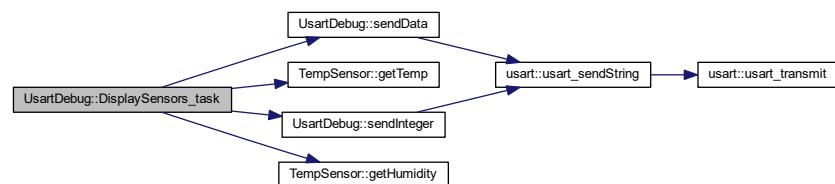
This task sends sensors data (temperature and humidity) on usart link every 5 seconds

Returns

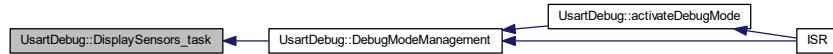
Nothing

Definition at line 69 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.19.3.5 isDebugModeActive()

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

Check is debug mode is active or not.

This function checks if debug mode is active or not.

Returns

TRUE is debug mode is active, FALSE otherwise

Definition at line 121 of file debug.cpp.

Here is the caller graph for this function:



3.19.3.6 sendBool()

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

Send a boolean data on USART link.

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

Parameters

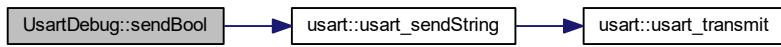
in	<i>data</i>	boolean data to be sent
----	-------------	-------------------------

Returns

Nothing

Definition at line 57 of file debug.cpp.

Here is the call graph for this function:

**3.19.3.7 sendData()**

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

Send a string on USART link.

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

Parameters

in	<i>str</i>	Pointer to the string being sent
----	------------	----------------------------------

Returns

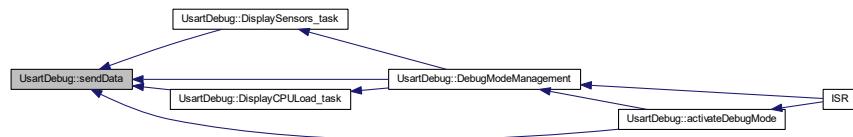
Nothing

Definition at line 37 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.19.3.8 sendInteger()

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

Send a integer data on USART link.

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

Parameters

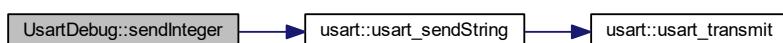
in	data	integer data to be sent
in	base	numerical base used to convert integer into string (between 2 and 36)

Returns

Nothing

Definition at line 43 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.19.4 Member Data Documentation

3.19.4.1 debug_state

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

Definition at line 116 of file debug.h.

3.19.4.2 debugModeActive_F

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

Debug mode activation flag

Definition at line 117 of file debug.h.

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

- [debug.h](#)
- [debug.cpp](#)

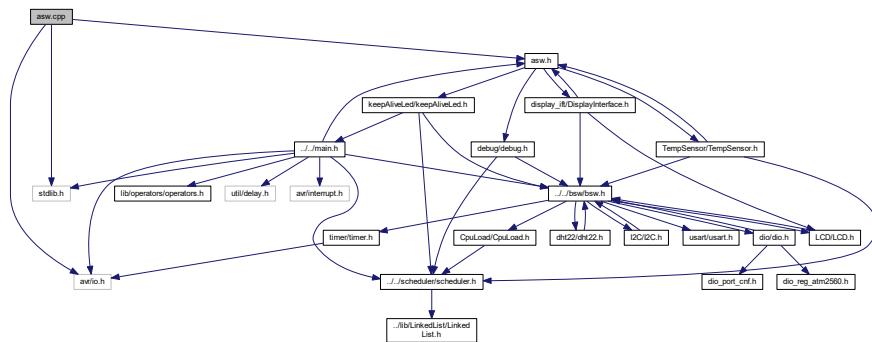
Chapter 4

File Documentation

4.1 asw.cpp File Reference

ASW main file.

```
#include <stdlib.h>
#include <avr/io.h>
#include "asw.h"
Include dependency graph for asw.cpp:
```



Functions

- void [asw_init \(\)](#)

Initialization of ASW.

Variables

- [T_ASW_cnf_struct ASW_cnf_struct](#)

4.1.1 Detailed Description

ASW main file.

Date

15 mars 2018

Author

nicls67

4.1.2 Function Documentation

4.1.2.1 asw_init()

```
void asw_init ( )
```

Initialization of ASW.

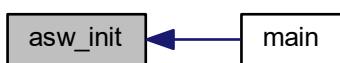
This function instantiates all applicative objects. The addresses of objects are then stored in ASW_cnf_struct structure. This function shall be called after BSW initialization function.

Returns

Nothing

Definition at line 19 of file asw.cpp.

Here is the caller graph for this function:



4.1.3 Variable Documentation

4.1.3.1 ASW_cnf_struct

`T_ASW_cnf_struct` ASW_cnf_struct

ASW configuration structure

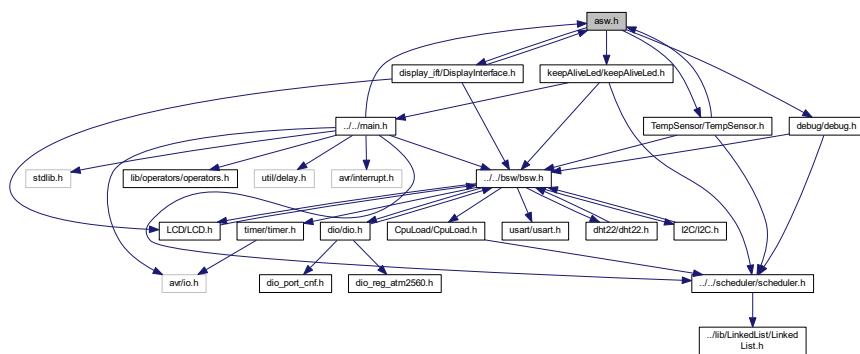
Definition at line 16 of file asw.cpp.

4.2 asw.h File Reference

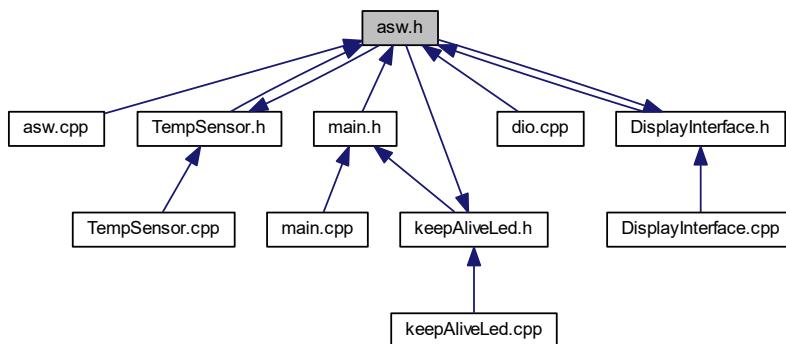
ASW main header file.

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

Include dependency graph for asw.h:



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



Classes

- struct [T_ASW_cnf_struct](#)
ASW configuration structure.

Functions

- void [asw_init\(\)](#)
Initialization of ASW.

Variables

- [T_ASW_cnf_struct ASW_cnf_struct](#)

4.2.1 Detailed Description

ASW main header file.

Date

15 mars 2018

Author

nicls67

4.2.2 Function Documentation

4.2.2.1 [asw_init\(\)](#)

`void asw_init()`

Initialization of ASW.

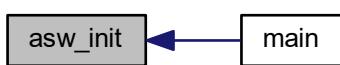
This function instantiates all applicative objects. The addresses of objects are then stored in `ASW_cnf_struct` structure. This function shall be called after BSW initialization function.

Returns

Nothing

Definition at line 19 of file `asw.cpp`.

Here is the caller graph for this function:



4.2.3 Variable Documentation

4.2.3.1 ASW_cnf_struct

`T_ASW_cnf_struct ASW_cnf_struct`

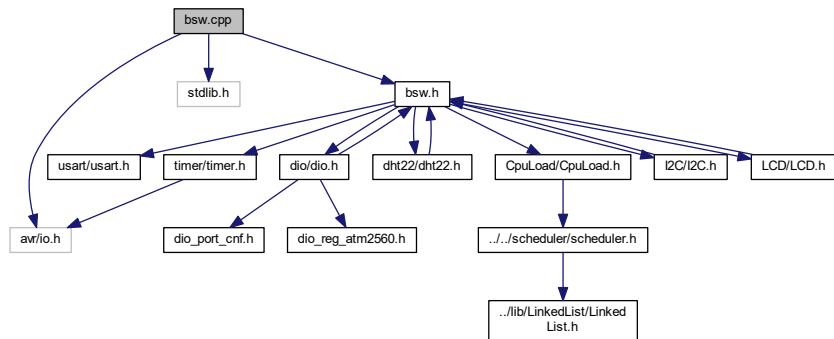
ASW configuration structure

Definition at line 16 of file asw.cpp.

4.3 bsw.cpp File Reference

BSW main file.

```
#include <avr/io.h>
#include <stdlib.h>
#include "bsw.h"
Include dependency graph for bsw.cpp:
```



Functions

- void `bsw_init ()`

Initialization of BSW.

Variables

- `T_BSW_cnf_struct BSW_cnf_struct`

4.3.1 Detailed Description

BSW main file.

Date

13 mars 2018

Author

nicls67

4.3.2 Function Documentation

4.3.2.1 bsw_init()

```
void bsw_init ( )
```

Initialization of BSW.

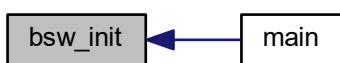
This function instantiates all driver objects, leading hardware initialization. The addresses of driver objects are then stored in BSW_cnf_struct structure.

Returns

Nothing

Definition at line 18 of file bsw.cpp.

Here is the caller graph for this function:



4.3.3 Variable Documentation

4.3.3.1 BSW_cnf_struct

`T_BSW_cnf_struct` BSW_cnf_struct

BSW configuration structure

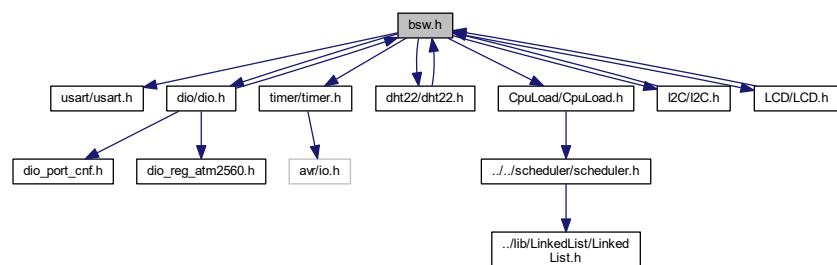
Definition at line 16 of file bsw.cpp.

4.4 bsw.h File Reference

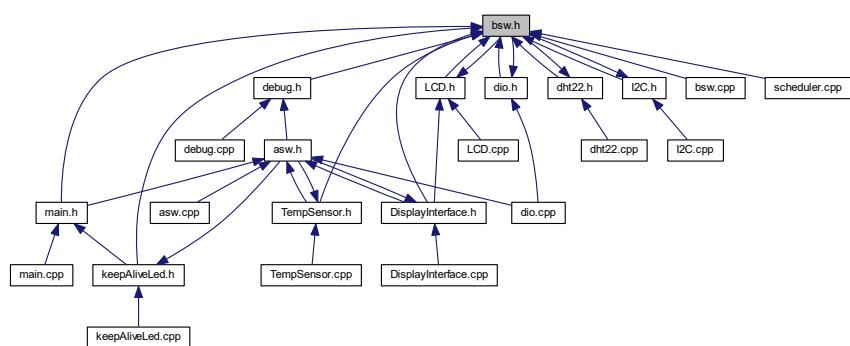
BSW main header file.

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

Include dependency graph for bsw.h:



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



Classes

- struct [T_BSW_cnf_struct](#)
BSW configuration structure.

Macros

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

Functions

- void [bsw_init](#) ()
Initialization of BSW.

Variables

- [T_BSW_cnf_struct BSW_cnf_struct](#)

4.4.1 Detailed Description

BSW main header file.

Date

13 mars 2018

Author

nicls67

4.4.2 Macro Definition Documentation

4.4.2.1 I2C_BITRATE

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

[I2C bus bitrate is 100 kHz](#)

Definition at line 27 of file bsw.h.

4.4.2.2 USART_BAUDRATE

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

uart connection to PC uses a baud rate of 9600

Definition at line 26 of file bsw.h.

4.4.3 Function Documentation

4.4.3.1 bsw_init()

```
void bsw_init ( )
```

Initialization of BSW.

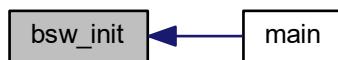
This function instantiates all driver objects, leading hardware initialization. The addresses of driver objects are then stored in BSW_cnf_struct structure.

Returns

Nothing

Definition at line 18 of file bsw.cpp.

Here is the caller graph for this function:



4.4.4 Variable Documentation

4.4.4.1 BSW_cnf_struct

```
T_BSW_cnf_struct BSW_cnf_struct
```

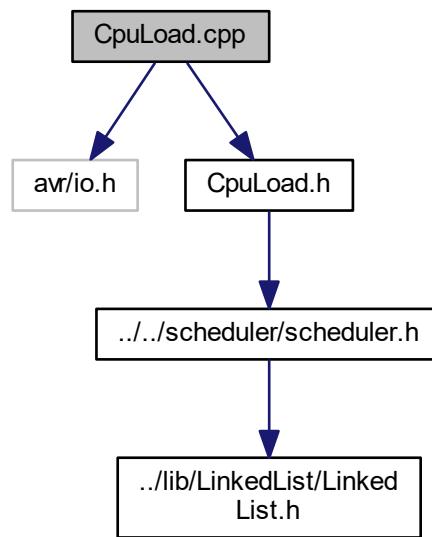
BSW configuration structure

Definition at line 16 of file bsw.cpp.

4.5 CpuLoad.cpp File Reference

Defines functions of class [CpuLoad](#).

```
#include <avr/io.h>
#include "CpuLoad.h"
Include dependency graph for CpuLoad.cpp:
```



4.5.1 Detailed Description

Defines functions of class [CpuLoad](#).

Date

21 mars 2019

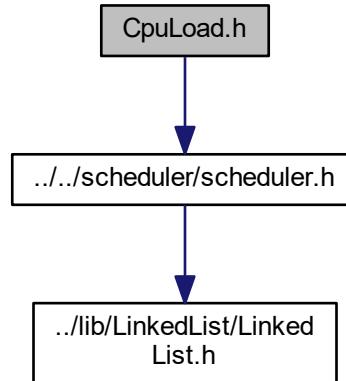
Author

nicls67

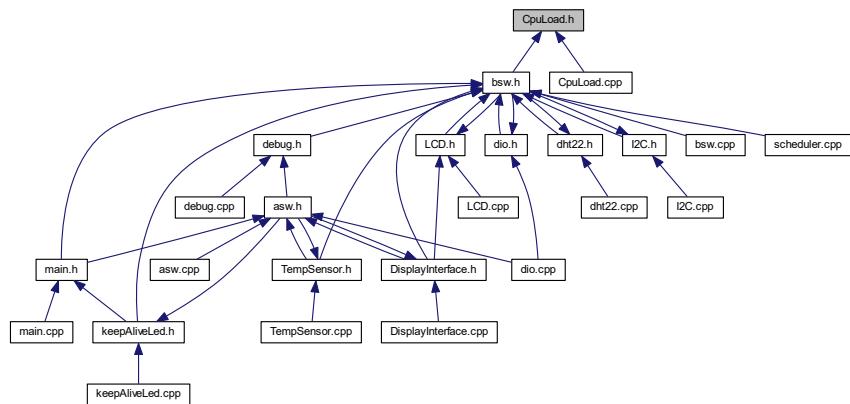
4.6 CpuLoad.h File Reference

[CpuLoad](#) class header file.

```
#include "../scheduler/scheduler.h"
Include dependency graph for CpuLoad.h:
```



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



Classes

- class [CpuLoad](#)

Class defining CPU load libraries.

Macros

- `#define NB_OF_SAMPLES 50`

4.6.1 Detailed Description

[CpuLoad](#) class header file.

Date

21 mars 2019

Author

nicls67

4.6.2 Macro Definition Documentation

4.6.2.1 NB_OF_SAMPLES

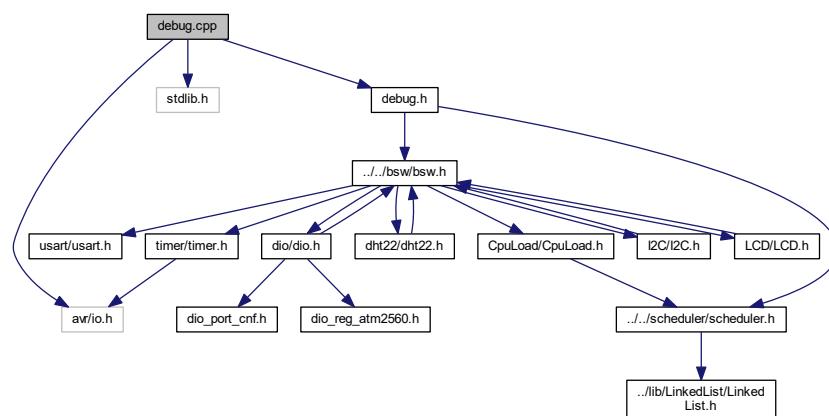
```
#define NB_OF_SAMPLES 50
```

Definition at line 13 of file CpuLoad.h.

4.7 debug.cpp File Reference

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

```
#include <avr/io.h>
#include <stdlib.h>
#include "debug.h"
Include dependency graph for debug.cpp:
```



Variables

- const char str_debug_main_menu []

Main menu of debug mode.

4.7.1 Detailed Description

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

Date

15 mars 2018

Author

nicls67

4.7.2 Variable Documentation

4.7.2.1 str_debug_main_menu

```
const char str_debug_main_menu[ ]
```

Initial value:

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

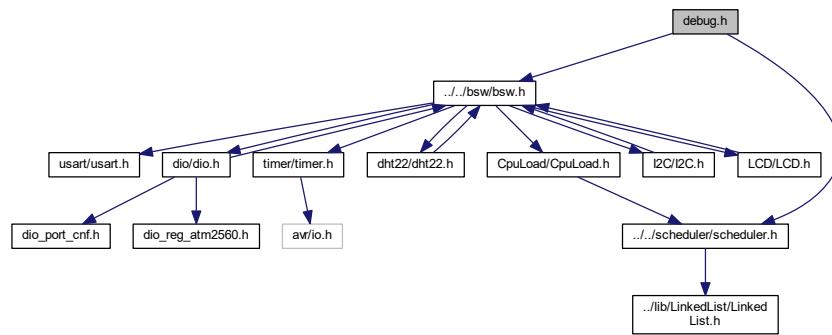
Main menu of debug mode.

Definition at line 20 of file debug.cpp.

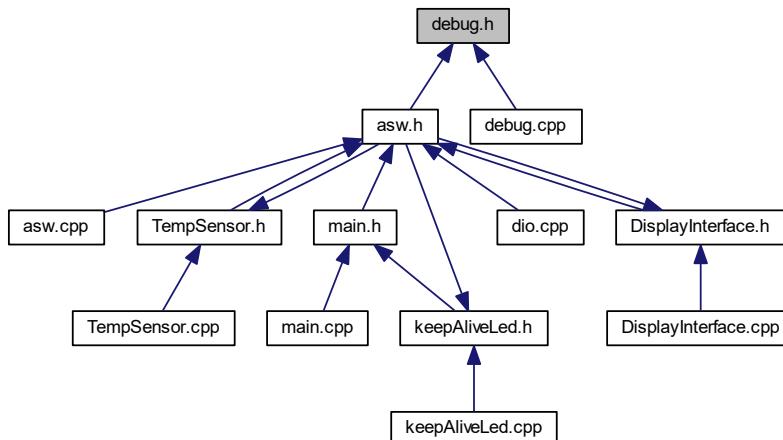
4.8 debug.h File Reference

Header file for debug and logging functions.

```
#include "../../bsw/bsw.h"
#include "../../scheduler/scheduler.h"
Include dependency graph for debug.h:
```



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



Classes

- class [UsartDebug](#)

Class used for debugging on usart link.

Macros

- `#define PERIOD_MS_TASK_DISPLAY_SENSORS 5000`
- `#define PERIOD_MS_TASK_DISPLAY_CPU_LOAD 5000`

Enumerations

- enum `debug_state_t` { `INIT`, `WAIT_INIT`, `DISPLAY_DATA`, `DISPLAY_CPU_LOAD` }
Defines the debug states.

4.8.1 Detailed Description

Header file for debug and logging functions.

Date

15 mars 2018

Author

nicls67

4.8.2 Macro Definition Documentation

4.8.2.1 PERIOD_MS_TASK_DISPLAY_CPU_LOAD

```
#define PERIOD_MS_TASK_DISPLAY_CPU_LOAD 5000
```

Period for displaying CPU load data

Definition at line 14 of file debug.h.

4.8.2.2 PERIOD_MS_TASK_DISPLAY_SENSORS

```
#define PERIOD_MS_TASK_DISPLAY_SENSORS 5000
```

Period for displaying temperature and humidity data

Definition at line 13 of file debug.h.

4.8.3 Enumeration Type Documentation

4.8.3.1 debug_state_t

```
enum debug_state_t
```

Defines the debug states.

Enumerator

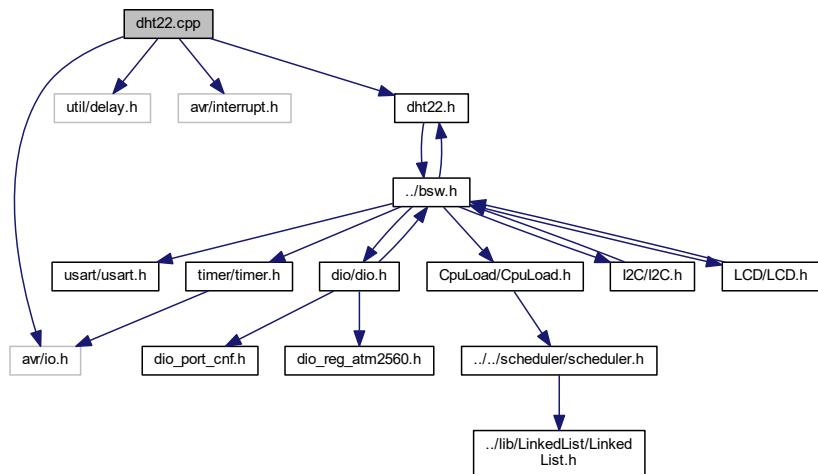
INIT	Init state : display the main menu
WAIT_INIT	Wait for a received character in init state
DISPLAY_DATA	Display sensor data in continuous
DISPLAY_CPU_LOAD	Display CPU load in continuous

Definition at line 20 of file debug.h.

4.9 dht22.cpp File Reference

This file defines classes for DHT22 driver.

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "dht22.h"
Include dependency graph for dht22.cpp:
```



Macros

- #define MAX_WAIT_TIME_US 100

4.9.1 Detailed Description

This file defines classes for DHT22 driver.

Date

23 mars 2018

Author

nicls67

4.9.2 Macro Definition Documentation

4.9.2.1 MAX_WAIT_TIME_US

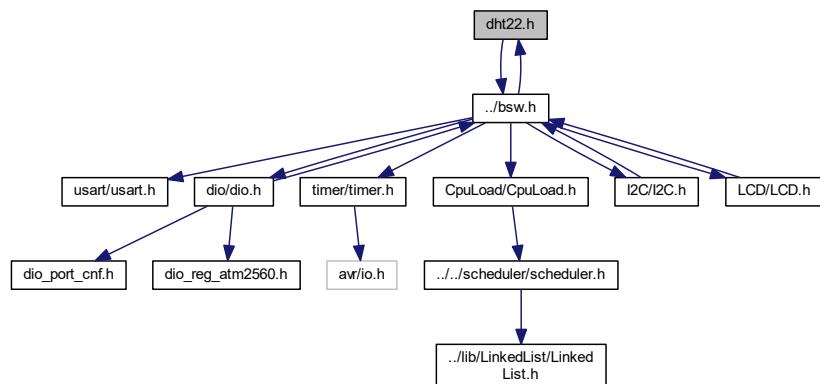
```
#define MAX_WAIT_TIME_US 100
```

Definition at line 20 of file dht22.cpp.

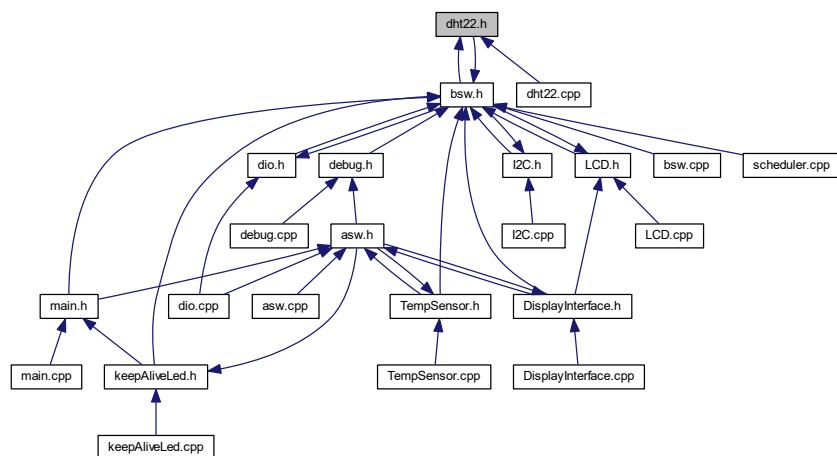
4.10 dht22.h File Reference

DHT22 driver header file.

```
#include "../bsw.h"
Include dependency graph for dht22.h:
```



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



Classes

- class `dht22`

DHT 22 driver class.

Macros

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

4.10.1 Detailed Description

DHT22 driver header file.

Date

23 mars 2018

Author

nicls67

4.10.2 Macro Definition Documentation

4.10.2.1 DHT22_PORT

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

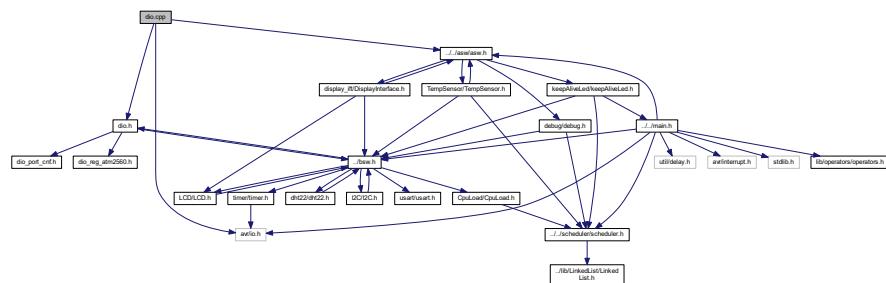
DHT22 is connected to port PB6

Definition at line 16 of file dht22.h.

4.11 dio.cpp File Reference

DIO library.

```
#include <avr/io.h>
#include "dio.h"
#include "../../../asw/asw.h"
Include dependency graph for dio.cpp:
```



4.11.1 Detailed Description

DIO library.

Date

13 mars 2018

Author

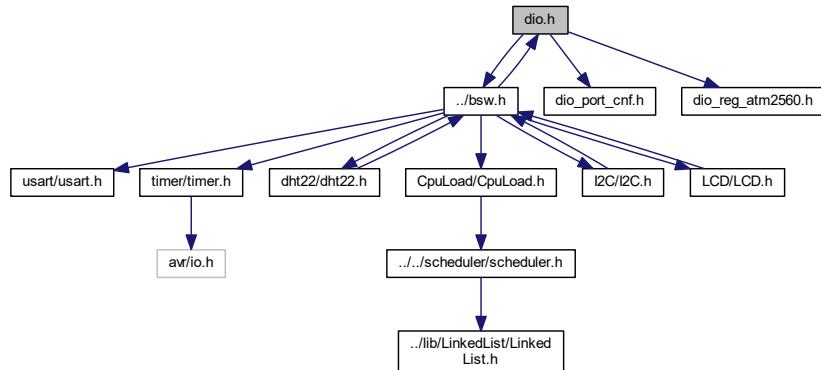
nicls67

4.12 dio.h File Reference

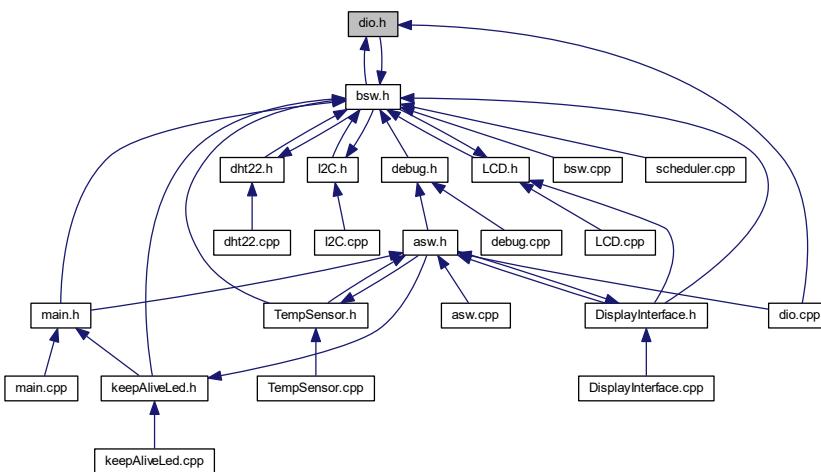
DIO library header file.

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

Include dependency graph for dio.h:



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



Classes

- class [dio](#)
DIO class.

Macros

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

4.12.1 Detailed Description

DIO library header file.

Date

13 mars 2018

Author

nicls67

4.12.2 Macro Definition Documentation

4.12.2.1 DECODE_PIN

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

Macro used to extract pin index

Definition at line 19 of file dio.h.

4.12.2.2 DECODE_PORT

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

Macro used to extract port index

Definition at line 18 of file dio.h.

4.12.2.3 ENCODE_PORT

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

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

Definition at line 17 of file dio.h.

4.12.2.4 PORT_A

```
#define PORT_A 0
```

PORTA index

Definition at line 21 of file dio.h.

4.12.2.5 PORT_B

```
#define PORT_B 1
```

PORTB index

Definition at line 22 of file dio.h.

4.12.2.6 PORT_C

```
#define PORT_C 2
```

PORTC index

Definition at line 23 of file dio.h.

4.12.2.7 PORT_CNF_IN

```
#define PORT_CNF_IN 0
```

Pin is configured as input

Definition at line 15 of file dio.h.

4.12.2.8 PORT_CNF_OUT

```
#define PORT_CNF_OUT 1
```

Pin is configured as output

Definition at line 14 of file dio.h.

4.12.2.9 PORT_D

```
#define PORT_D 3
```

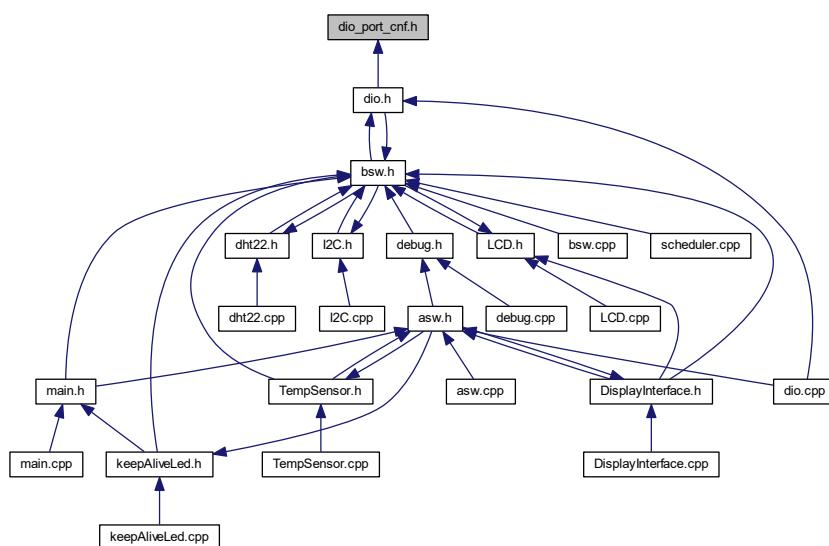
PORTD index

Definition at line 24 of file dio.h.

4.13 dio_port_cnf.h File Reference

Digital ports configuration file.

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



Macros

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

4.13.1 Detailed Description

Digital ports configuration file.

Date

19 mars 2019

Author

nicls67

4.13.2 Macro Definition Documentation

4.13.2.1 PORTB_CNF_DDRB

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

Defines the configuration of DDRB register.

This constant defines the direction of IO pins of PORT B. It will configure register DDRB.

PB0 : N/A
PB1 : N/A
PB2 : N/A
PB3 : N/A
PB4 : N/A
PB5 : N/A
PB6 : OUT
PB7 : OUT

Definition at line 25 of file dio_port_cnf.h.

4.13.2.2 PORTB_CNF_PORTB

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

Defines the configuration of PORTB register.

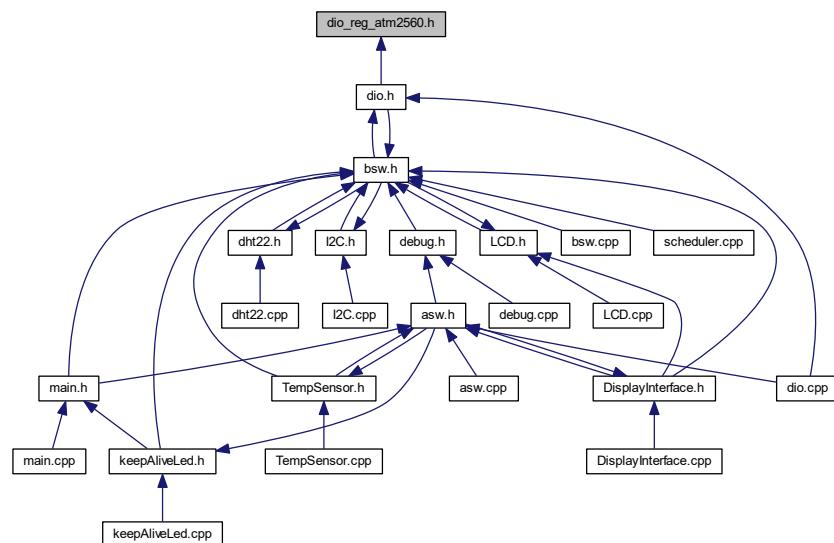
This constant defines the initial value of IO pins for PORT B. It will configure register PORTB. Pins configured as input shall not be configured here.

PB0 : N/A
 PB1 : N/A
 PB2 : N/A
 PB3 : N/A
 PB4 : N/A
 PB5 : N/A
 PB6 : HIGH
 PB7 : HIGH

Definition at line 40 of file dio_port_cnf.h.

4.14 dio_reg_atm2560.h File Reference

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



Macros

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

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

4.14.1 Macro Definition Documentation

4.14.1.1 DDRA_PTR

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

Macro defining pointer to DDR A register

Definition at line 24 of file dio_reg_atm2560.h.

4.14.1.2 DDRB_PTR

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

Macro defining pointer to DDR B register

Definition at line 25 of file dio_reg_atm2560.h.

4.14.1.3 DDRC_PTR

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

Macro defining pointer to DDR C register

Definition at line 26 of file dio_reg_atm2560.h.

4.14.1.4 DDRD_PTR

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

Macro defining pointer to DDR D register

Definition at line 27 of file dio_reg_atm2560.h.

4.14.1.5 PINA_PTR

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

Macro defining pointer to PIN A register

Definition at line 19 of file dio_reg_atm2560.h.

4.14.1.6 PINB_PTR

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

Macro defining pointer to PIN B register

Definition at line 20 of file dio_reg_atm2560.h.

4.14.1.7 PINC_PTR

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

Macro defining pointer to PIN C register

Definition at line 21 of file dio_reg_atm2560.h.

4.14.1.8 PIND_PTR

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

Macro defining pointer to PIN D register

Definition at line 22 of file dio_reg_atm2560.h.

4.14.1.9 PORTA_PTR

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

Macro defining pointer to PORT A register

Definition at line 14 of file dio_reg_atm2560.h.

4.14.1.10 PORTB_PTR

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

Macro defining pointer to PORT B register

Definition at line 15 of file dio_reg_atm2560.h.

4.14.1.11 PORTC_PTR

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

Macro defining pointer to PORT C register

Definition at line 16 of file dio_reg_atm2560.h.

4.14.1.12 PORTD_PTR

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

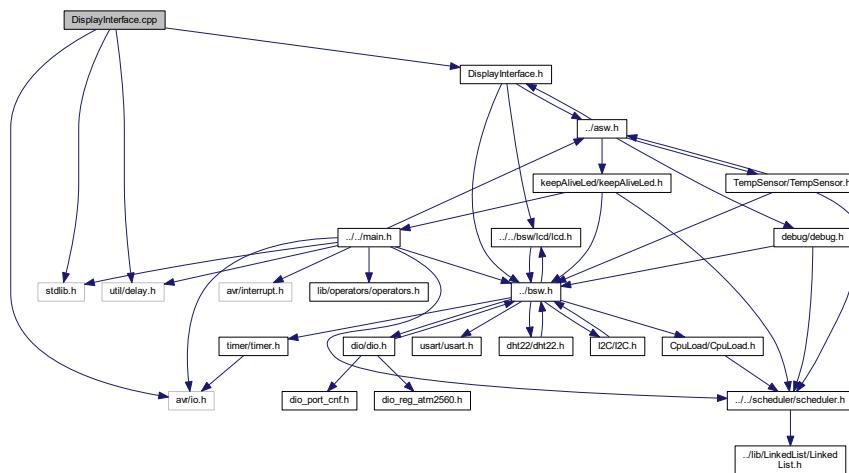
Macro defining pointer to PORT D register

Definition at line 17 of file dio_reg_atm2560.h.

4.15 DisplayInterface.cpp File Reference

Source code file for display services.

```
#include <stdlib.h>
#include <avr/io.h>
#include <util/delay.h>
#include "DisplayInterface.h"
Include dependency graph for DisplayInterface.cpp:
```



4.15.1 Detailed Description

Source code file for display services.

Date

23 avr. 2019

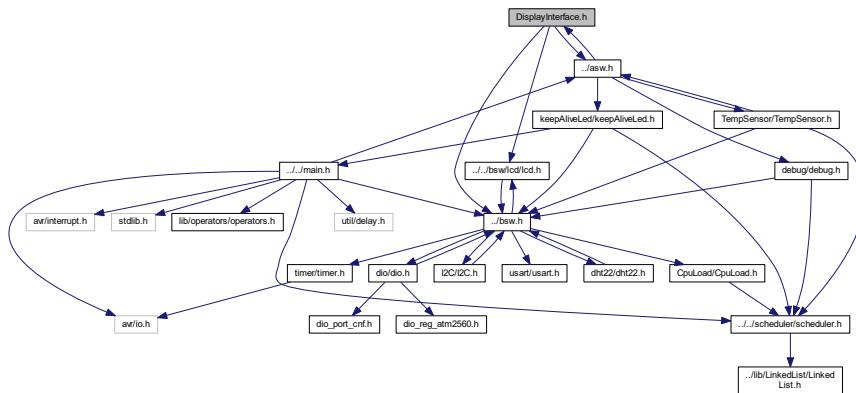
Author

nicls67

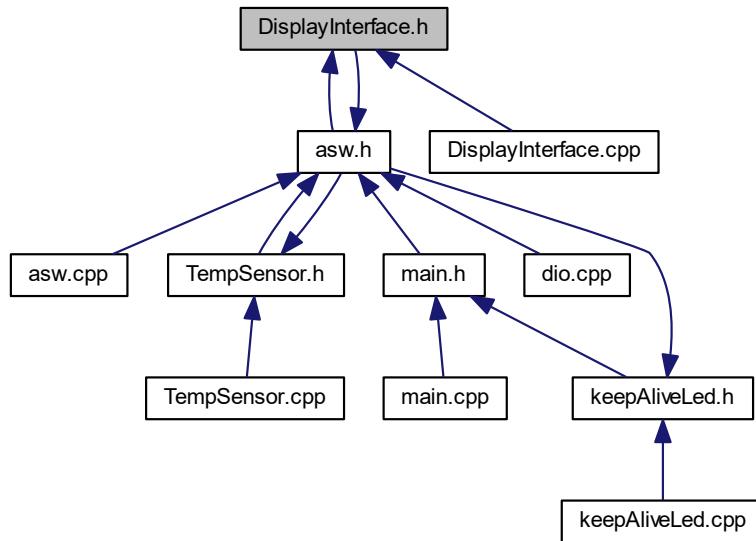
4.16 DisplayInterface.h File Reference

[DisplayInterface](#) class header file.

```
#include "../../bsw/lcd/lcd.h"
#include "../../bsw/bsw.h"
#include "../asw.h"
Include dependency graph for DisplayInterface.h:
```



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



Classes

- struct [T_Display_shift_data](#)
Structure containing shift 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.

Variables

- const [T_LCD_conf_struct](#) [LCD_init_cnf](#)
LCD configuration structure.

4.16.1 Detailed Description

[DisplayInterface](#) class header file.

Date

23 avr. 2019

Author

nicls67

4.16.2 Macro Definition Documentation

4.16.2.1 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 60 of file DisplayInterface.h.

4.16.2.2 DISPLAY_LINE_SHIFT_TEMPO_TIME

```
#define DISPLAY_LINE_SHIFT_TEMPO_TIME 6
```

In "line shift" mode for line display, a temporization of 3 periods is added at the end and the beginning of the lines

Definition at line 61 of file DisplayInterface.h.

4.16.3 Enumeration Type Documentation

4.16.3.1 T_DisplayInterface_LineDisplayMode

```
enum T_DisplayInterface_LineDisplayMode
```

Modes for line display.

This enumeration defines the available modes for the line display functionality :

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

Enumerator

NORMAL	
LINE_SHIFT	
GO_TO_NEXT_LINE	

Definition at line 38 of file DisplayInterface.h.

4.16.4 Variable Documentation

4.16.4.1 LCD_init_cnf

```
const T_LCD_conf_struct LCD_init_cnf
```

Initial value:

```
= {  
    0x27,  
    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.

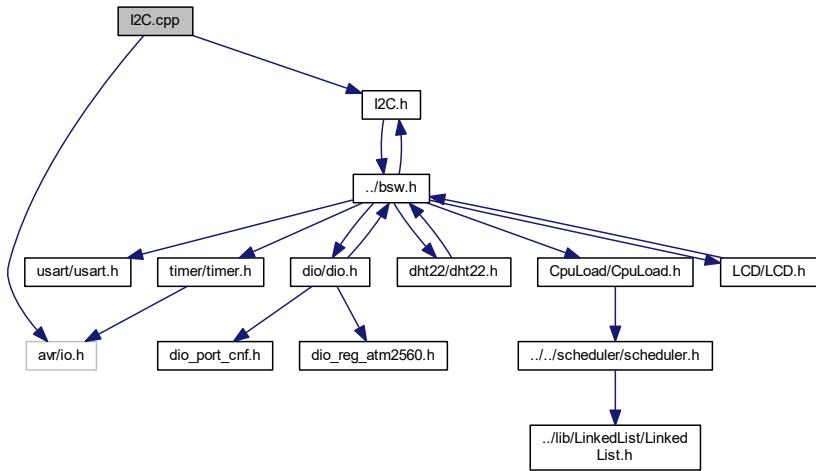
This structure defines the initial configuration of the LCD screen.

Definition at line 19 of file DisplayInterface.h.

4.17 I2C.cpp File Reference

Two-wire interface (I2C) source file.

```
#include <avr/io.h>
#include "I2C.h"
Include dependency graph for I2C.cpp:
```



4.17.1 Detailed Description

Two-wire interface ([I2C](#)) source file.

Date

19 avr. 2019

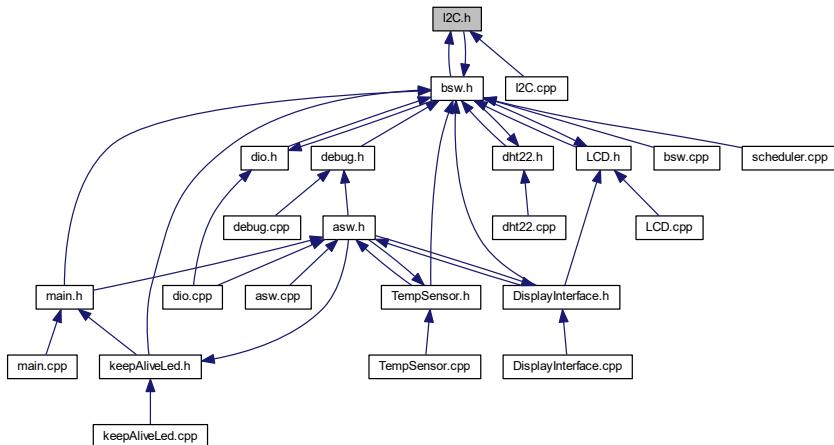
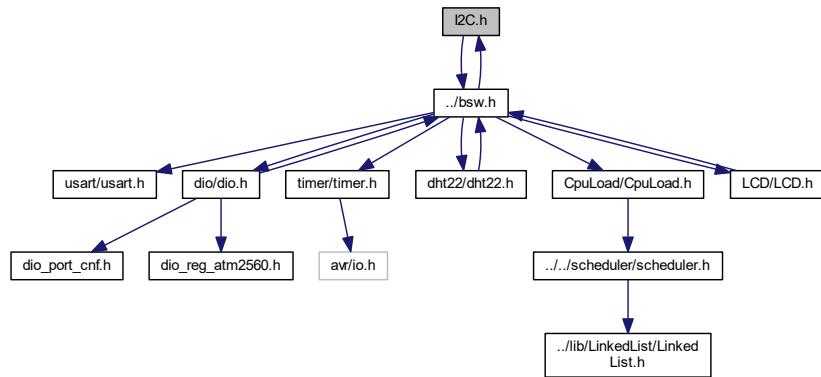
Author

nicls67

4.18 I2C.h File Reference

[I2C](#) class header file.

```
#include "../bsw.h"
Include dependency graph for I2C.h:
```



Classes

- class [I2C](#)

Two-wire serial interface ([I2C](#)) class definition.

Macros

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

4.18.1 Detailed Description

I2C class header file.

Date

19 avr. 2019

Author

nicls67

4.18.2 Macro Definition Documentation

4.18.2.1 DATA_ACK

```
#define DATA_ACK 0x28
```

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

Definition at line 15 of file I2C.h.

4.18.2.2 SLA_ACK

```
#define SLA_ACK 0x18
```

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

Definition at line 14 of file I2C.h.

4.18.2.3 START

```
#define START 0x08
```

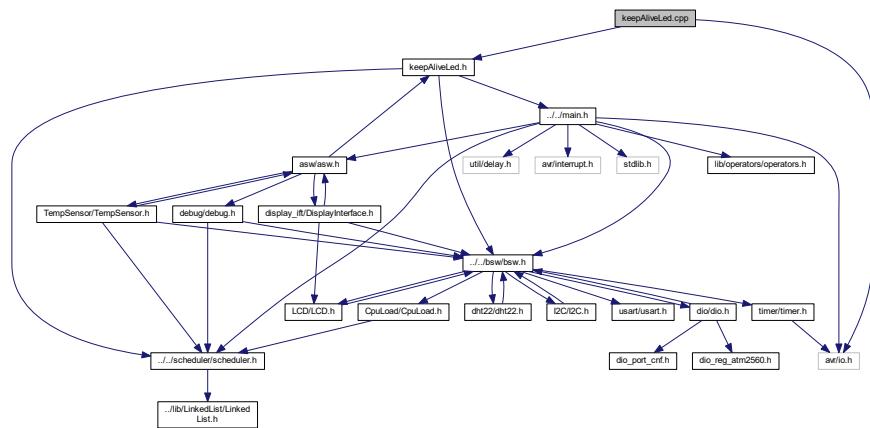
TWSR status code : START condition transmitted

Definition at line 13 of file I2C.h.

4.19 keepAliveLed.cpp File Reference

Definition of function for class [keepAliveLed](#).

```
#include <avr/io.h>
#include "keepAliveLed.h"
Include dependency graph for keepAliveLed.cpp:
```



4.19.1 Detailed Description

Definition of function for class [keepAliveLed](#).

Date

17 mars 2018

Author

nicls67

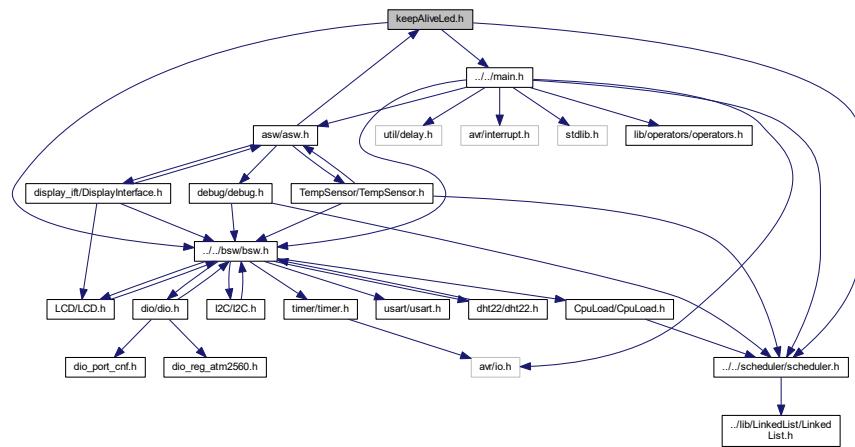
4.20 keepAliveLed.h File Reference

Class [keepAliveLed](#) header file.

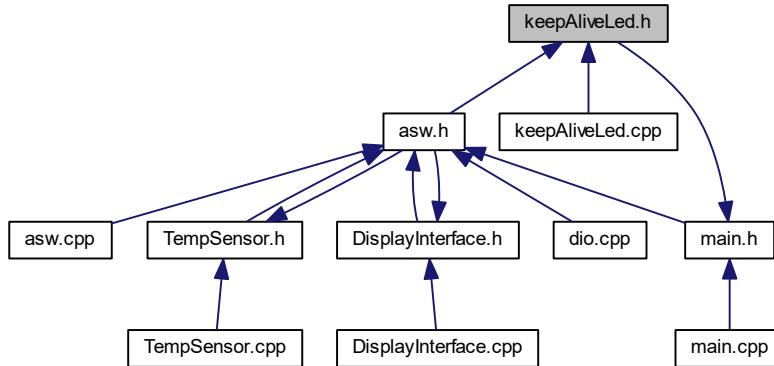
```
#include "../../bsw/bsw.h"
#include "../../scheduler/scheduler.h"
```

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

Include dependency graph for keepAliveLed.h:



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



Classes

- class `keepAliveLed`

Class for keep-alive LED blinking.

Macros

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

4.20.1 Detailed Description

Class [keepAliveLed](#) header file.

Date

17 mars 2018

Author

nicls67

4.20.2 Macro Definition Documentation

4.20.2.1 LED_PORT

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

LED is connected to port PB7

Definition at line 16 of file [keepAliveLed.h](#).

4.20.2.2 PERIOD_MS_TASK_LED

```
#define PERIOD_MS_TASK_LED SW_PERIOD_MS
```

Period for led blinking

Definition at line 15 of file [keepAliveLed.h](#).

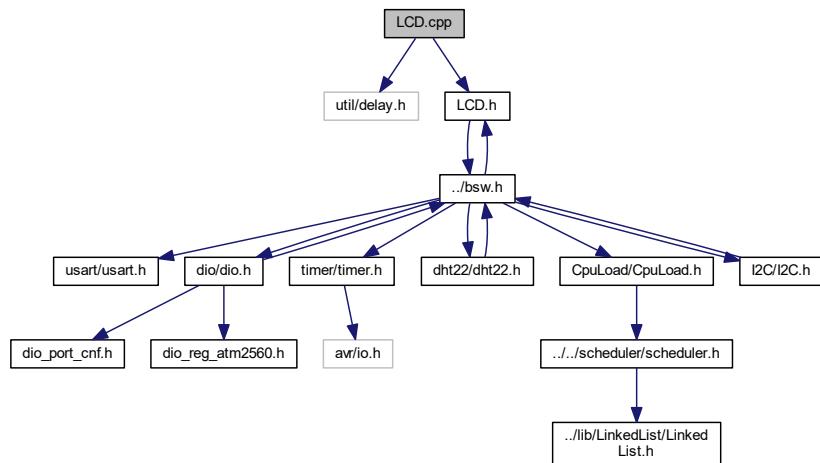
4.21 LCD.cpp File Reference

[LCD](#) class source file.

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

```
#include "LCD.h"
```

Include dependency graph for [LCD.cpp](#):



4.21.1 Detailed Description

[LCD](#) class source file.

Date

20 avr. 2019

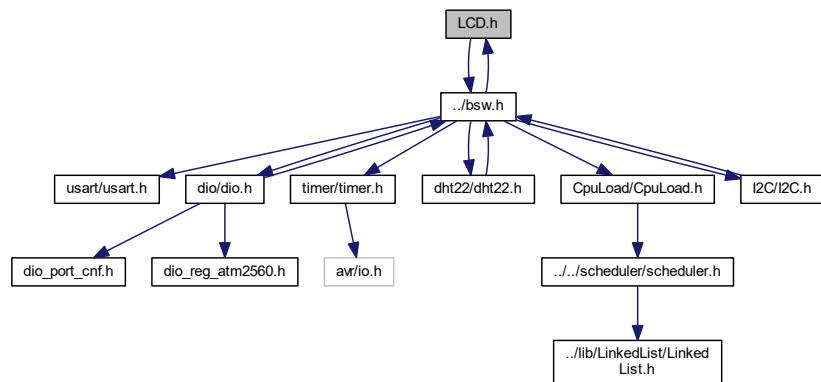
Author

nicls67

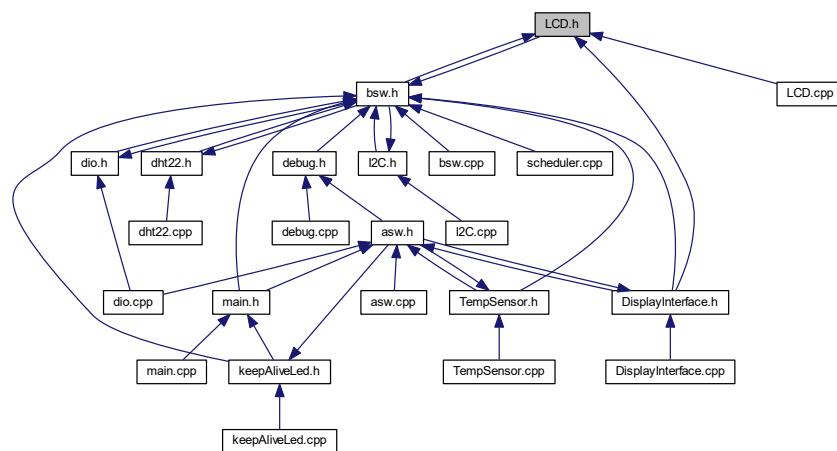
4.22 LCD.h File Reference

[LCD](#) class header file.

```
#include "../bsw.h"
Include dependency graph for LCD.h:
```



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



Classes

- struct `T_LCD_conf_struct`
- class `LCD`

Class for `LCD S2004A` display driver.

Macros

- `#define EN_PIN 2`
- `#define RW_PIN 1`
- `#define RS_PIN 0`
- `#define BACKLIGHT_PIN 3`
- `#define LCD_INST_CLR_DISPLAY_BIT 0`
- `#define LCD_INST_FUNCTION_SET 5`
- `#define LCD_INST_DISPLAY_CTRL 3`
- `#define LCD_INST_ENTRY_MODE_SET 2`
- `#define LCD_INST_SET_DDRAM_ADDR 7`
- `#define LCD_FCT_SET_FIELD_DL 4`
- `#define LCD_FCT_SET_FIELD_N 3`
- `#define LCD_FCT_SET_FIELD_F 2`
- `#define LCD_DISPLAY_CTRL_FIELD_D 2`
- `#define LCD_DISPLAY_CTRL_FIELD_C 1`
- `#define LCD_DISPLAY_CTRL_FIELD_B 0`
- `#define LCD_CNF_SHIFT_ID 1`
- `#define LCD_CNF_SHIFT_SH 0`
- `#define LCD_CNF_ONE_LINE 0`
- `#define LCD_CNF_TWO_LINE 1`
- `#define LCD_CNF_FONT_5_8 0`
- `#define LCD_CNF_FONT_5_11 1`
- `#define LCD_CNF_DISPLAY_ON 1`
- `#define LCD_CNF_DISPLAY_OFF 0`
- `#define LCD_CNF_CURSOR_ON 1`
- `#define LCD_CNF_CURSOR_OFF 0`
- `#define LCD_CNF_CURSOR_BLINK_ON 1`
- `#define LCD_CNF_CURSOR_BLINK_OFF 0`
- `#define LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT 1`
- `#define LCD_CNF_ENTRY_MODE_DIRECTION_LEFT 0`
- `#define LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON 1`
- `#define LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF 0`
- `#define LCD_CNF_BACKLIGHT_ON 1`
- `#define LCD_CNF_BACKLIGHT_OFF 0`
- `#define LCD_RAM_1_LINE_MIN 0`
- `#define LCD_RAM_1_LINE_MAX 0x4F`
- `#define LCD_RAM_2_LINES_MIN_1 0`
- `#define LCD_RAM_2_LINES_MAX_1 0x27`
- `#define LCD_RAM_2_LINES_MIN_2 0x40`
- `#define LCD_RAM_2_LINES_MAX_2 0x67`
- `#define LCD_WAIT_CLR_RETURN 1600`
- `#define LCD_WAIT_OTHER_MODES 40`
- `#define LCD_SIZE_NB_CHAR_PER_LINE 20`
- `#define LCD_SIZE_NB_LINES 4`

Enumerations

- enum `T_LCD_command` {
 `LCD_CMD_FUNCTION_SET`, `LCD_CMD_CLEAR_DISPLAY`, `LCD_CMD_DISPLAY_CTRL`, `LCD_CMD_ENTRY_MODE_SET`,
 `LCD_CMD_SET_DDRAM_ADDR` }
 LCD commands enumeration.
- enum `T_LCD_config_mode` { `LCD_MODE_INSTRUCTION` = 0, `LCD_MODE_DATA` = 1 }
 LCD modes enumeration.
- enum `T_LCD_ram_area` { `LCD_DATA_DDRAM`, `LCD_DATA_CGRAM` }
 Screen RAM definition.

4.22.1 Detailed Description

`LCD` class header file.

Date

20 avr. 2019

Author

nicls67

4.22.2 Macro Definition Documentation

4.22.2.1 BACKLIGHT_PIN

```
#define BACKLIGHT_PIN 3
```

Backlight pin is on P3

Definition at line 17 of file LCD.h.

4.22.2.2 EN_PIN

```
#define EN_PIN 2
```

EN bit is on P2

Definition at line 14 of file LCD.h.

4.22.2.3 LCD_CNF_BACKLIGHT_OFF

```
#define LCD_CNF_BACKLIGHT_OFF 0
```

Backlight is disabled

Definition at line 70 of file LCD.h.

4.22.2.4 LCD_CNF_BACKLIGHT_ON

```
#define LCD_CNF_BACKLIGHT_ON 1
```

Backlight is enabled

Definition at line 69 of file LCD.h.

4.22.2.5 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.

4.22.2.6 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.

4.22.2.7 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.

4.22.2.8 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.

4.22.2.9 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.

4.22.2.10 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.

4.22.2.11 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.

4.22.2.12 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.

4.22.2.13 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.

4.22.2.14 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.

4.22.2.15 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.

4.22.2.16 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.

4.22.2.17 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.

4.22.2.18 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.

4.22.2.19 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.

4.22.2.20 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.

4.22.2.21 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.

4.22.2.22 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.

4.22.2.23 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.

4.22.2.24 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.

4.22.2.25 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.

4.22.2.26 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.

4.22.2.27 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.

4.22.2.28 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.

4.22.2.29 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.

4.22.2.30 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.

4.22.2.31 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.

4.22.2.32 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.

4.22.2.33 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.

4.22.2.34 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.

4.22.2.35 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.

4.22.2.36 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.

4.22.2.37 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.

4.22.2.38 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.

4.22.2.39 LCD_SIZE_NB_LINES

```
#define LCD_SIZE_NB_LINES 4
```

LCD screen has 4 lines

Definition at line 86 of file LCD.h.

4.22.2.40 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.

4.22.2.41 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.

4.22.2.42 RS_PIN

```
#define RS_PIN 0
```

RS pin is on P0

Definition at line 16 of file LCD.h.

4.22.2.43 RW_PIN

```
#define RW_PIN 1
```

RW pin is on P1

Definition at line 15 of file LCD.h.

4.22.3 Enumeration Type Documentation

4.22.3.1 T_LCD_command

```
enum T_LCD_command
```

LCD commands enumeration.

This enumeration defines all command modes available for [LCD](#) configuration

Enumerator

LCD_CMD_FUNCTION_SET	
LCD_CMD_CLEAR_DISPLAY	
LCD_CMD_DISPLAY_CTRL	
LCD_CMD_ENTRY_MODE_SET	
LCD_CMD_SET_DDRAM_ADDR	

Definition at line 93 of file LCD.h.

4.22.3.2 T_LCD_config_mode

```
enum T_LCD_config_mode
```

LCD modes enumeration.

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

Enumerator

LCD_MODE_INSTRUCTION	
LCD_MODE_DATA	

Definition at line 107 of file LCD.h.

4.22.3.3 T_LCD_ram_area

```
enum T_LCD_ram_area
```

Screen RAM definition.

This enumeration defines the RAM areas of the [LCD](#) screen : DDRAM for display, CGRAM for characters generation

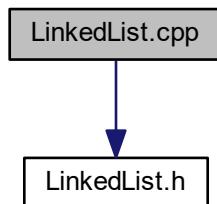
Enumerator

LCD_DATA_DDRAM	
LCD_DATA_CGRAM	

Definition at line 118 of file LCD.h.

4.23 LinkedList.cpp File Reference

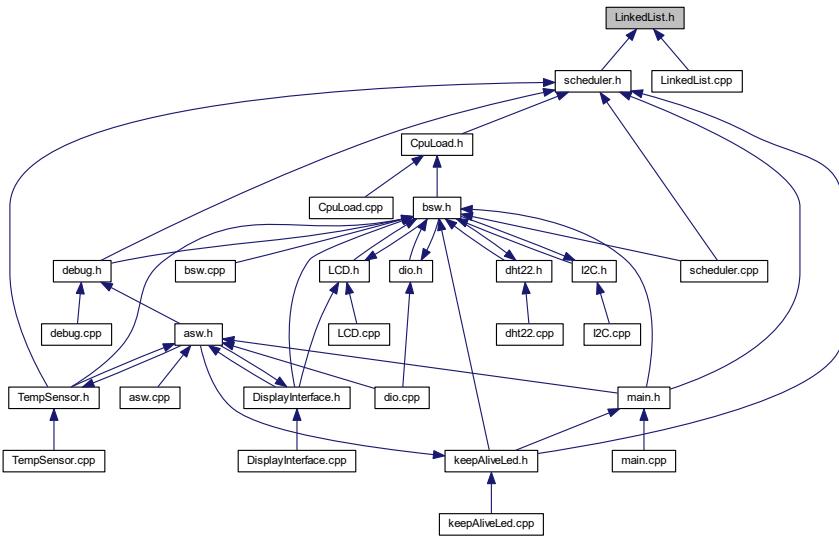
```
#include "LinkedList.h"  
Include dependency graph for LinkedList.cpp:
```



4.24 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`)

4.24.1 Detailed Description

Linked List library header file.

Date

27 avr. 2019

Author

nicls67

4.24.2 Typedef Documentation

4.24.2.1 CompareFctPtr_t

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

Type defining a pointer to the comparison function

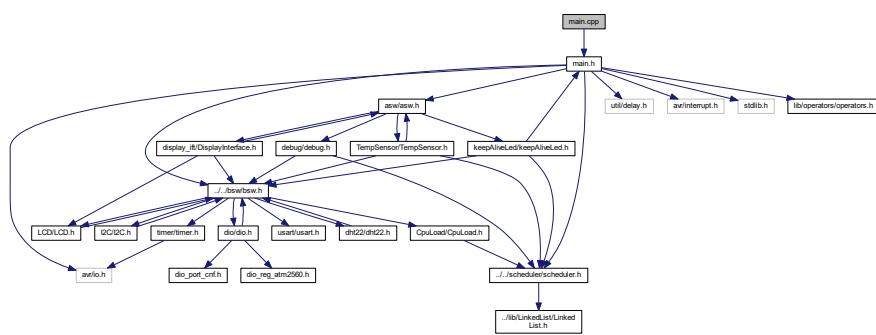
Definition at line 14 of file LinkedList.h.

4.25 main.cpp File Reference

Background task file.

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

Include dependency graph for main.cpp:



Functions

- **ISR (TIMER1_COMPA_vect)**
Main software interrupt.
- **ISR (USART0_RX_vect)**
USART Rx Complete interrupt.
- int **main (void)**
Background task of program.

4.25.1 Detailed Description

Background task file.

Date

12 mars 2018

Author

nicls67

4.25.2 Function Documentation

4.25.2.1 ISR() [1/2]

```
ISR (
    TIMER1_COMPA_vect )
```

Main software interrupt.

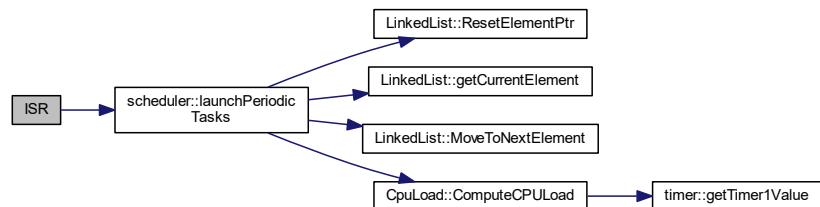
This function handles the interrupt raised by Timer #1. It wakes up the software every 500 ms to perform applications.

Returns

Nothing

Definition at line 19 of file main.cpp.

Here is the call graph for this function:



4.25.2.2 ISR() [2/2]

```
ISR (
    USART0_RX_vect )
```

USART Rx Complete interrupt.

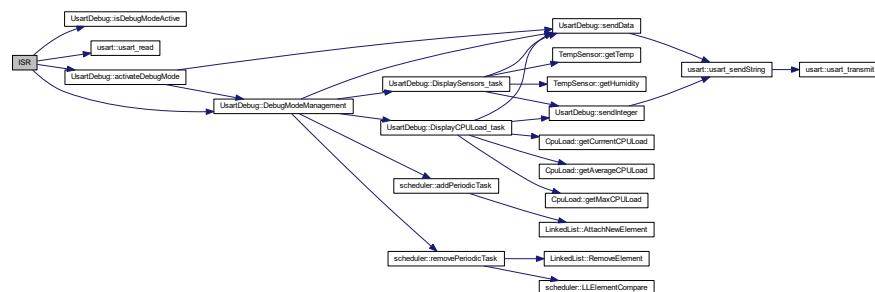
This function handles the interrupt raised when a frame has been received by USART. If debug mode mode is active, it calls debug mode management function. If inactive, it calls debug mode activation function if the received character is 'a'

Returns

Nothing

Definition at line 31 of file main.cpp.

Here is the call graph for this function:

**4.25.2.3 main()**

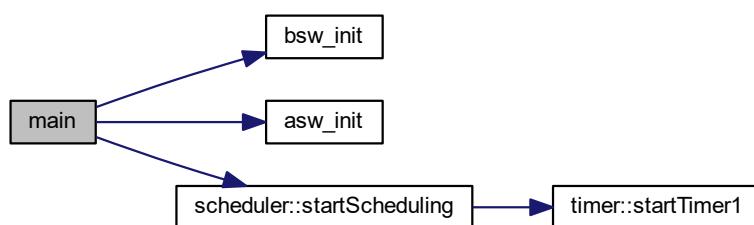
```
int main (
    void )
```

Background task of program.

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

Definition at line 51 of file main.cpp.

Here is the call graph for this function:

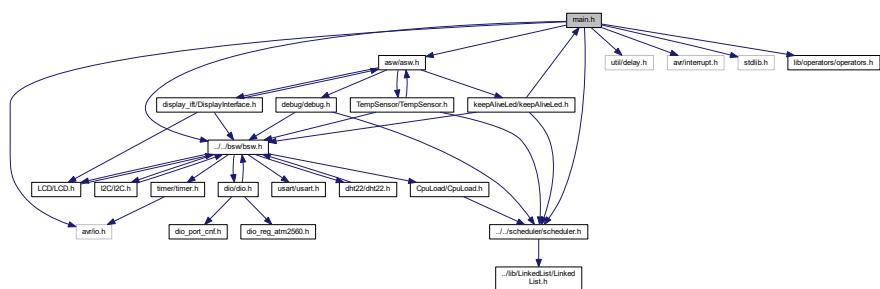


4.26 main.h File Reference

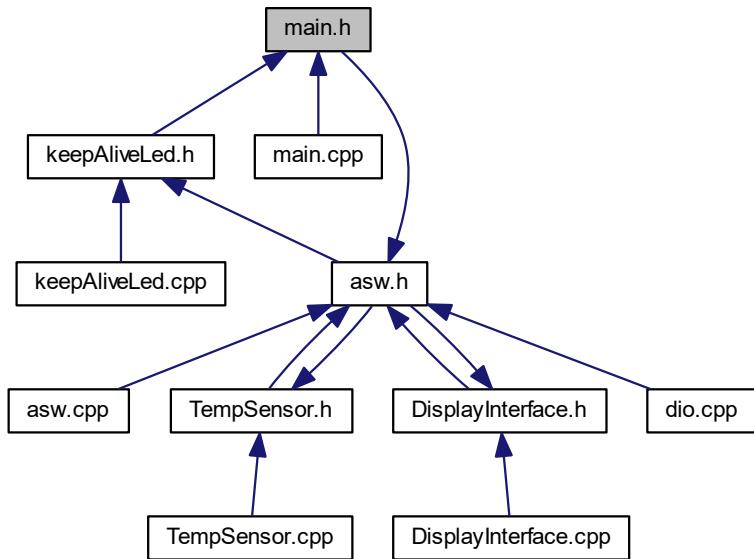
Background task header file.

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

Include dependency graph for main.h:



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



4.26.1 Detailed Description

Background task header file.

Date

17 mars 2018

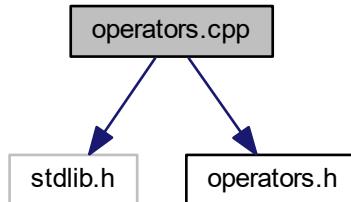
Author

nicls67

4.27 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.

4.27.1 Detailed Description

c++ operators definitions

Date

14 mars 2018

Author

nicls67

4.27.2 Function Documentation

4.27.2.1 operator delete()

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

Operator delete.

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

Parameters

in	<i>ptr</i>	Pointer to the start of memory zone to free
----	------------	---

Returns

Nothing

Definition at line 18 of file operators.cpp.

4.27.2.2 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	<i>a_size</i>	memory size to allocate
----	---------------	-------------------------

Returns

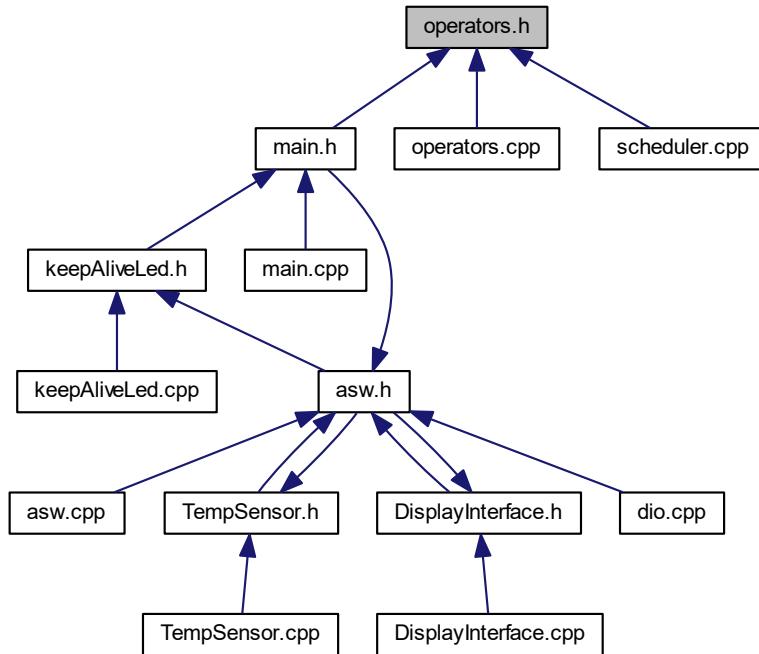
Pointer to the start of allocated memory zone

Definition at line 13 of file operators.cpp.

4.28 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.

4.28.1 Detailed Description

c++ operators definitions header file

Date

14 mars 2018

Author

nicls67

4.28.2 Function Documentation

4.28.2.1 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.

4.28.2.2 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.

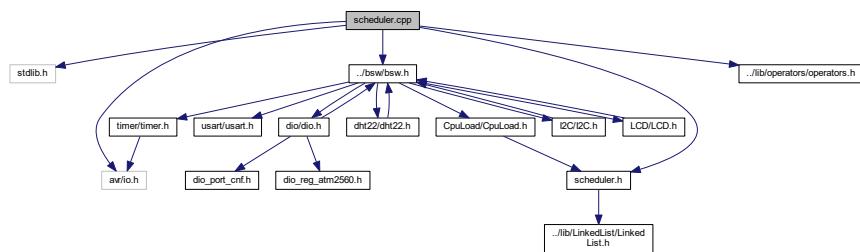
4.29 scheduler.cpp File Reference

Defines scheduler class.

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

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

Include dependency graph for scheduler.cpp:



Variables

- `scheduler * p_scheduler`

4.29.1 Detailed Description

Defines scheduler class.

Date

16 mars 2018

Author

nicls67

4.29.2 Variable Documentation

4.29.2.1 p_scheduler

`scheduler* p_scheduler`

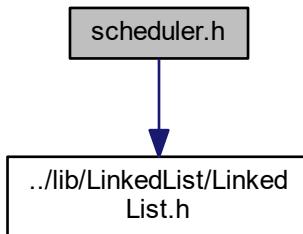
Pointer to scheduler object

Definition at line 18 of file scheduler.cpp.

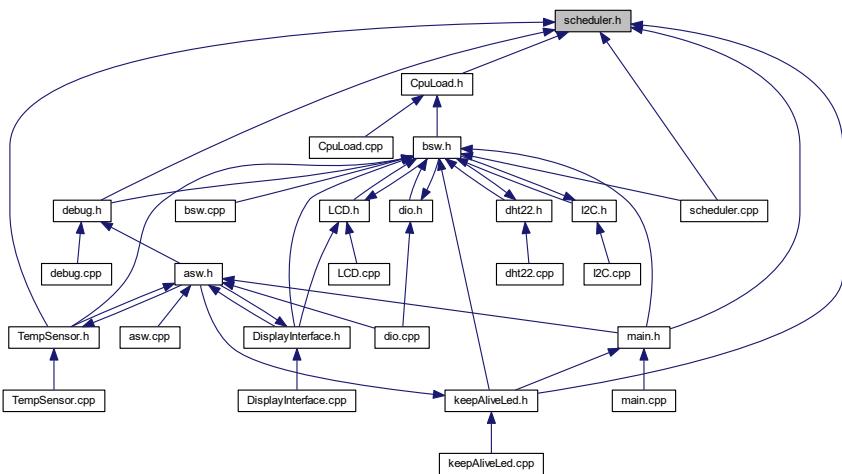
4.30 scheduler.h File Reference

Scheduler class header file.

```
#include "../lib/LinkedList/LinkedList.h"
Include dependency graph for scheduler.h:
```



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_scheduler`

4.30.1 Detailed Description

Scheduler class header file.

Date

16 mars 2018

Author

nicls67

4.30.2 Macro Definition Documentation

4.30.2.1 PRESCALER_PERIODIC_TIMER

```
#define PRESCALER_PERIODIC_TIMER 256
```

Value of prescaler to use for periodic timer

Definition at line 17 of file scheduler.h.

4.30.2.2 SW_PERIOD_MS

```
#define SW_PERIOD_MS 500
```

Software period, used to define periodic timer interrupt

Definition at line 16 of file scheduler.h.

4.30.2.3 TIMER_CTC_VALUE

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

Compare value for periodic timer

Definition at line 18 of file scheduler.h.

4.30.3 Typedef Documentation

4.30.3.1 TaskPtr_t

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

Type defining a pointer to function.

Definition at line 23 of file scheduler.h.

4.30.4 Variable Documentation

4.30.4.1 p_scheduler

```
scheduler* p_scheduler
```

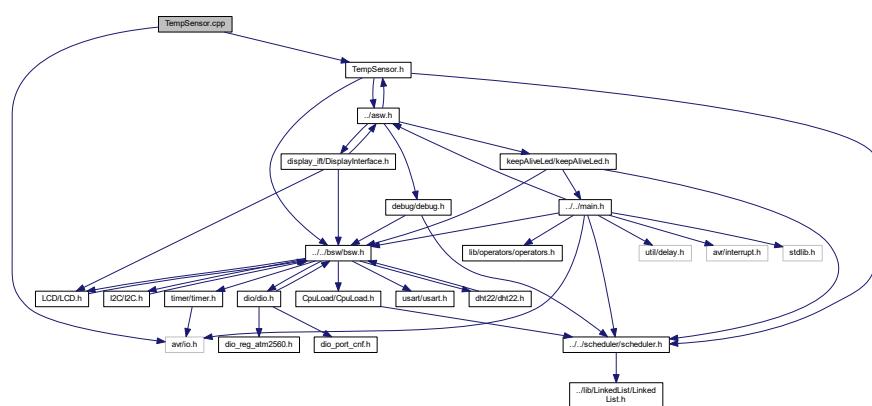
Pointer to scheduler object

Definition at line 18 of file scheduler.cpp.

4.31 TempSensor.cpp File Reference

Defines function of class [TempSensor](#).

```
#include <avr/io.h>
#include "TempSensor.h"
Include dependency graph for TempSensor.cpp:
```



Macros

- `#define PIT_BEFORE_INVALID 60`

4.31.1 Detailed Description

Defines function of class [TempSensor](#).

Date

23 mars 2018

Author

nicls67

4.31.2 Macro Definition Documentation

4.31.2.1 PIT_BEFORE_INVALID

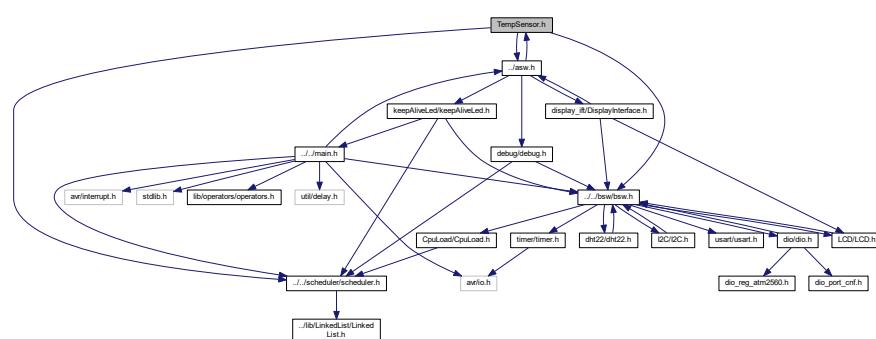
```
#define PIT_BEFORE_INVALID 60
```

Definition at line 14 of file [TempSensor.cpp](#).

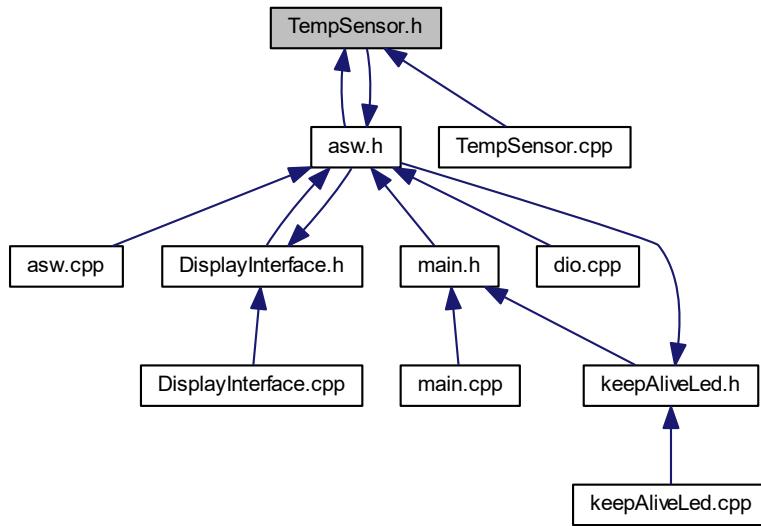
4.32 TempSensor.h File Reference

Class [TempSensor](#) header file.

```
#include "../../scheduler/scheduler.h"
#include "../../bsw/bsw.h"
#include "../asw.h"
Include dependency graph for TempSensor.h:
```



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



Classes

- class [TempSensor](#)
Class for temperature sensor.

Macros

- `#define PERIOD_MS_TASK_TEMP_SENSOR 5000`

4.32.1 Detailed Description

Class [TempSensor](#) header file.

Date

23 mars 2018

Author

nicls67

4.32.2 Macro Definition Documentation

4.32.2.1 PERIOD_MS_TASK_TEMP_SENSOR

```
#define PERIOD_MS_TASK_TEMP_SENSOR 5000
```

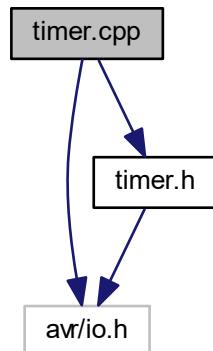
Period for reading temperature data

Definition at line 13 of file TempSensor.h.

4.33 timer.cpp File Reference

Defines function for class timer.

```
#include <avr/io.h>
#include "timer.h"
Include dependency graph for timer.cpp:
```



4.33.1 Detailed Description

Defines function for class timer.

Date

15 mars 2018

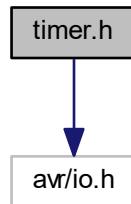
Author

nicls67

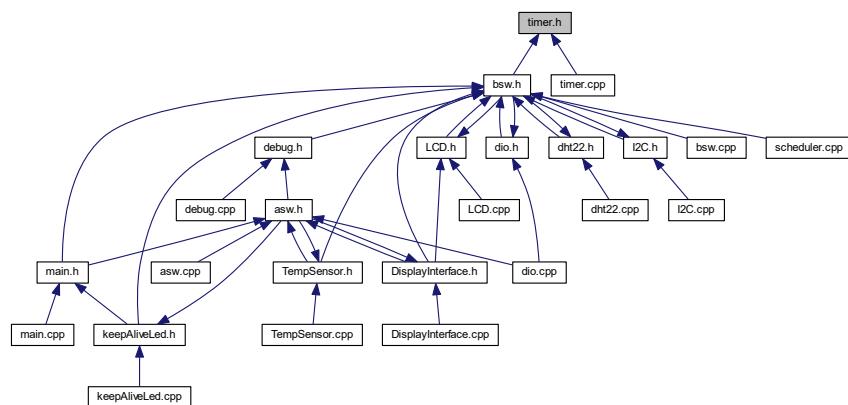
4.34 timer.h File Reference

Timer class header file.

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



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



Classes

- class [timer](#)

Class defining a timer.

4.34.1 Detailed Description

Timer class header file.

Date

15 mars 2018

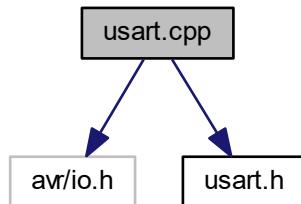
Author

nicls67

4.35 usart.cpp File Reference

BSW library for USART.

```
#include <avr/io.h>
#include "usart.h"
Include dependency graph for usart.cpp:
```



4.35.1 Detailed Description

BSW library for USART.

Date

13 mars 2018

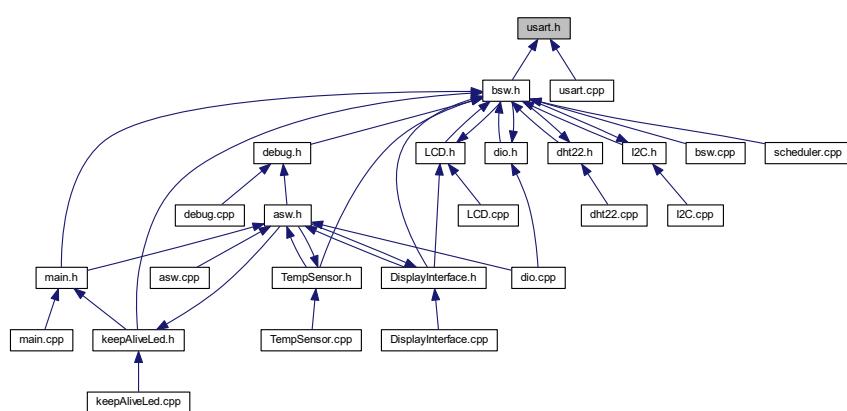
Author

nicls67

4.36 usart.h File Reference

Header file for USART library.

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



Classes

- class [uart](#)
USART serial bus class.

4.36.1 Detailed Description

Header file for USART library.

Date

13 mars 2018

Author

nicls67

Index

ASW_cnf_struct
 asw.cpp, 102
 asw.h, 105
activateDebugMode
 UsartDebug, 92
addPeriodicTask
 scheduler, 60
asw.cpp, 101
 ASW_cnf_struct, 102
 asw_init, 102
asw.h, 103
 ASW_cnf_struct, 105
 asw_init, 104
asw_init
 asw.cpp, 102
 asw.h, 104
AttachNewElement
 LinkedList, 53
avg_load
 CpuLoad, 8

BACKLIGHT_PIN
 LCD.h, 140
BSW_cnf_struct
 bsw.cpp, 106
 bsw.h, 109
backlight_en
 T_LCD_conf_struct, 72
backlight_enable
 LCD, 50
BaudRate
 uart, 91
bitrate
 I2C, 33
blinkLed_task
 keepAliveLed, 35
bsw.cpp, 105
 BSW_cnf_struct, 106
 bsw_init, 106
bsw.h, 107
 BSW_cnf_struct, 109
 bsw_init, 109
 I2C_BITRATE, 108
 USART_BAUDRATE, 108
bsw_init
 bsw.cpp, 106
 bsw.h, 109

ClearLine
 DisplayInterface, 24

cnfCursorBlink
 LCD, 50
cnfCursorOnOff
 LCD, 50
cnfDisplayOnOff
 LCD, 50
cnfEntryModeDir
 LCD, 50
cnfEntryModeShift
 LCD, 50
cnfFontType
 LCD, 51
cnfI2C_addr
 LCD, 51
cnfLineNumber
 LCD, 51
command
 LCD, 38
CompareFctPtr_t
 LinkedList.h, 151
ComputeCPUload
 CpuLoad, 6
ConfigureBacklight
 LCD, 38
ConfigureCursorBlink
 LCD, 39
ConfigureCursorOnOff
 LCD, 40
ConfigureDisplayOnOff
 LCD, 40
ConfigureEntryModeDir
 LCD, 41
ConfigureEntryModeShift
 LCD, 42
ConfigureFontType
 LCD, 42
ConfigureI2CAddr
 LCD, 43
ConfigureLineNumber
 LCD, 44
configureTimer1
 timer, 84
CpuLoad, 5
 avg_load, 8
 ComputeCPUload, 6
 CpuLoad, 6
 current_load, 8
 getAverageCPUload, 6
 getCurrentCPUload, 7

getMaxCPUload, 7
 last_sum_value, 8
 max_load, 9
 sample_cnt, 9
 sample_idx, 9
 sample_mem, 9
 CpuLoad.cpp, 110
 CpuLoad.h, 110
 NB_OF_SAMPLES, 112
 curElement_ptr
 LinkedList, 57
 current_load
 CpuLoad, 8
 cursor_en
 T_LCD_conf_struct, 72
 cursorBlink_en
 T_LCD_conf_struct, 72

 DATA_ACK
 I2C.h, 134
 DDRA_PTR
 dio_reg_atm2560.h, 125
 DDRB_PTR
 dio_reg_atm2560.h, 125
 DDRC_PTR
 dio_reg_atm2560.h, 125
 DDRD_PTR
 dio_reg_atm2560.h, 125
 DECODE_PIN
 dio.h, 120
 DECODE_PORT
 dio.h, 120
 DHT22_PORT
 dht22.h, 118
 DISPLAY_LINE_SHIFT_PERIOD_MS
 DisplayInterface.h, 130
 DISPLAY_LINE_SHIFT_TEMPO_TIME
 DisplayInterface.h, 130
 data_ptr
 LinkedList::T_LL_element, 74
 ddram_addr
 LCD, 51
 debug.cpp, 112
 str_debug_main_menu, 113
 debug.h, 114
 debug_state_t, 115
 PERIOD_MS_TASK_DISPLAY_CPU_LOAD, 115
 PERIOD_MS_TASK_DISPLAY_SENSORS, 115
 debug_state
 UsartDebug, 99
 debug_state_t
 debug.h, 115
 debugModeActive_F
 UsartDebug, 99
 DebugModeManagement
 UsartDebug, 93
 dht22, 10
 dht22, 10
 initializeCommunication, 11

 read, 11
 dht22.cpp, 116
 MAX_WAIT_TIME_US, 117
 dht22.h, 117
 DHT22_PORT, 118
 dio, 12
 dio, 13
 dio_changePortPinCnf, 14
 dio_getPort, 15
 dio_getPort_fast, 16
 dio_invertPort, 16
 dio_memorizePINaddress, 17
 dio_setPort, 18
 getDDRxAddress, 19
 getPINxAddress, 19
 getPORTxAddress, 20
 PINx_addr_mem, 21
 PINx_idx_mem, 21
 ports_init, 21
 dio.cpp, 118
 dio.h, 119
 DECODE_PIN, 120
 DECODE_PORT, 120
 ENCODE_PORT, 121
 PORT_CNF_IN, 121
 PORT_CNF_OUT, 122
 PORT_A, 121
 PORT_B, 121
 PORT_C, 121
 PORT_D, 122
 dio_changePortPinCnf
 dio, 14
 dio_getPort
 dio, 15
 dio_getPort_fast
 dio, 16
 dio_invertPort
 dio, 16
 dio_memorizePINaddress
 dio, 17
 dio_port_cnf.h, 122
 PORTB_CNF_DDRB, 123
 PORTB_CNF_PORTB, 123
 dio_reg_atm2560.h, 124
 DDRA_PTR, 125
 DDRB_PTR, 125
 DDRC_PTR, 125
 DDRD_PTR, 125
 PINA_PTR, 125
 PINB_PTR, 126
 PINC_PTR, 126
 PIND_PTR, 126
 PORTA_PTR, 126
 PORTB_PTR, 126
 PORTC_PTR, 127
 PORTD_PTR, 127
 dio_setPort
 dio, 18

display_en
 T_LCD_conf_struct, 72

DisplayCPULoad_task
 UsartDebug, 94

DisplayFullLine
 DisplayInterface, 24

DisplayInterface, 22
 ClearLine, 24
 DisplayFullLine, 24
 DisplayInterface, 23
 dummy, 29
 FindFirstCharAddr, 25
 getLLShiftDataPtr, 26
 IsLineEmpty, 27
 LL_shift_data_ptr, 29
 LLElementCompare, 27
 lineEmptyTab, 29
 p_lcd, 29
 shiftLine_task, 28

DisplayInterface.cpp, 127

DisplayInterface.h, 128
 DISPLAY_LINE_SHIFT_PERIOD_MS, 130
 DISPLAY_LINE_SHIFT_TEMPO_TIME, 130
 LCD_init_cnf, 131
 T_DisplayInterface_LineDisplayMode, 130

DisplaySensors_task
 UsartDebug, 95

dummy
 DisplayInterface, 29

EN_PIN
 LCD.h, 140

ENCODE_PORT
 dio.h, 121

entryModeDir
 T_LCD_conf_struct, 72

entryModeShift
 T_LCD_conf_struct, 72

FindFirstCharAddr
 DisplayInterface, 25

firstElement
 LinkedList, 58

fontType_cnf
 T_LCD_conf_struct, 73

getAverageCPULoad
 CpuLoad, 6

getCurrentElement
 LinkedList, 54

getCurrentCPULoad
 CpuLoad, 7

GetDDRAMAddress
 LCD, 44

getDDRxAddress
 dio, 19

getHumPtr
 TempSensor, 78

getHumidity

 TempSensor, 77

getLLShiftDataPtr
 DisplayInterface, 26

GetLineNumberCnf
 LCD, 45

getMaxCPULoad
 CpuLoad, 7

getPINxAddress
 dio, 19

getPORTxAddress
 dio, 20

getPitNumber
 scheduler, 61

getTemp
 TempSensor, 78

getTempPtr
 TempSensor, 79

getTimer1Value
 timer, 85

I2C.cpp, 131

I2C.h, 132
 DATA_ACK, 134
 SLA_ACK, 134
 START, 134

I2C_BITRATE
 bsw.h, 108

I2C, 30
 bitrate, 33
 I2C, 30
 initializeBus, 31
 setBitRate, 31
 setTxAddress, 32
 tx_address, 33
 writeByte, 32

i2c_addr
 T_LCD_conf_struct, 73

ISR
 main.cpp, 153

initializeBus
 I2C, 31

initializeCommunication
 dht22, 11

InitializeScreen
 LCD, 45

isDebugModeActive
 UsartDebug, 96

IsLLEmpty
 LinkedList, 54

IsLineEmpty
 DisplayInterface, 27

keepAliveLed, 34
 blinkLed_task, 35
 keepAliveLed, 34

keepAliveLed.cpp, 135

keepAliveLed.h, 135
 LED_PORT, 137
 PERIOD_MS_TASK_LED, 137

LCD.cpp, 137
 LCD.h, 138
 BACKLIGHT_PIN, 140
 EN_PIN, 140
 LCD_CNF_BACKLIGHT_OFF, 140
 LCD_CNF_BACKLIGHT_ON, 141
 LCD_CNF_CURSOR_BLINK_OFF, 141
 LCD_CNF_CURSOR_BLINK_ON, 141
 LCD_CNF_CURSOR_OFF, 141
 LCD_CNF_CURSOR_ON, 141
 LCD_CNF_DISPLAY_OFF, 142
 LCD_CNF_DISPLAY_ON, 142
 LCD_CNF_ENTRY_MODE_DIRECTION_LEFT, 142
 LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT, 142
 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_↔_OFF, 142
 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_↔_ON, 143
 LCD_CNF_FONT_5_11, 143
 LCD_CNF_FONT_5_8, 143
 LCD_CNF_ONE_LINE, 143
 LCD_CNF_SHIFT_ID, 143
 LCD_CNF_SHIFT_SH, 144
 LCD_CNF_TWO_LINE, 144
 LCD_DISPLAY_CTRL_FIELD_B, 144
 LCD_DISPLAY_CTRL_FIELD_C, 144
 LCD_DISPLAY_CTRL_FIELD_D, 144
 LCD_FCT_SET_FIELD_DL, 145
 LCD_FCT_SET_FIELD_F, 145
 LCD_FCT_SET_FIELD_N, 145
 LCD_INST_CLR_DISPLAY_BIT, 145
 LCD_INST_DISPLAY_CTRL, 145
 LCD_INST_ENTRY_MODE_SET, 146
 LCD_INST_FUNCTION_SET, 146
 LCD_INST_SET_DDRAM_ADDR, 146
 LCD_RAM_1_LINE_MAX, 146
 LCD_RAM_1_LINE_MIN, 146
 LCD_RAM_2_LINES_MAX_1, 147
 LCD_RAM_2_LINES_MAX_2, 147
 LCD_RAM_2_LINES_MIN_1, 147
 LCD_RAM_2_LINES_MIN_2, 147
 LCD_SIZE_NB_CHAR_PER_LINE, 147
 LCD_SIZE_NB_LINES, 148
 LCD_WAIT_CLR_RETURN, 148
 LCD_WAIT_OTHER_MODES, 148
 RS_PIN, 148
 RW_PIN, 148
 T_LCD_command, 149
 T_LCD_config_mode, 149
 T_LCD_ram_area, 149
 LCD_CNF_BACKLIGHT_OFF
 LCD.h, 140
 LCD_CNF_BACKLIGHT_ON
 LCD.h, 141
 LCD_CNF_CURSOR_BLINK_OFF
 LCD.h, 141
 LCD_CNF_CURSOR_BLINK_ON
 LCD.h, 141
 LCD_CNF_CURSOR_OFF
 LCD.h, 141
 LCD_CNF_CURSOR_ON
 LCD.h, 141
 LCD_CNF_DISPLAY_OFF
 LCD.h, 142
 LCD_CNF_DISPLAY_ON
 LCD.h, 142
 LCD_CNF_ENTRY_MODE_DIRECTION_LEFT
 LCD.h, 142
 LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT
 LCD.h, 142
 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF
 LCD.h, 142
 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON
 LCD.h, 143
 LCD_CNF_FONT_5_11
 LCD.h, 143
 LCD_CNF_FONT_5_8
 LCD.h, 143
 LCD_CNF_ONE_LINE
 LCD.h, 143
 LCD_CNF_SHIFT_ID
 LCD.h, 143
 LCD_CNF_TWO_LINE
 LCD.h, 144
 LCD_DISPLAY_CTRL_FIELD_B
 LCD.h, 144
 LCD_DISPLAY_CTRL_FIELD_C
 LCD.h, 144
 LCD_DISPLAY_CTRL_FIELD_D
 LCD.h, 144
 LCD_FCT_SET_FIELD_DL
 LCD.h, 145
 LCD_FCT_SET_FIELD_F
 LCD.h, 145
 LCD_FCT_SET_FIELD_N
 LCD.h, 145
 LCD_INST_CLR_DISPLAY_BIT
 LCD.h, 145
 LCD_INST_DISPLAY_CTRL
 LCD.h, 145
 LCD_INST_ENTRY_MODE_SET
 LCD.h, 146
 LCD_INST_FUNCTION_SET
 LCD.h, 146
 LCD_INST_SET_DDRAM_ADDR
 LCD.h, 146
 LCD_RAM_1_LINE_MAX
 LCD.h, 146
 LCD_RAM_1_LINE_MIN
 LCD.h, 146
 LCD_RAM_2_LINES_MAX_1
 LCD.h, 147

LCD_RAM_2_LINES_MAX_2
LCD.h, 147

LCD_RAM_2_LINES_MIN_1
LCD.h, 147

LCD_RAM_2_LINES_MIN_2
LCD.h, 147

LCD_SIZE_NB_CHAR_PER_LINE
LCD.h, 147

LCD_SIZE_NB_LINES
LCD.h, 148

LCD_WAIT_CLR_RETURN
LCD.h, 148

LCD_WAIT_OTHER_MODES
LCD.h, 148

LCD_init_cnf
DisplayInterface.h, 131

LCD, 35

- backlight_enable, 50
- cnfCursorBlink, 50
- cnfCursorOnOff, 50
- cnfDisplayOnOff, 50
- cnfEntryModeDir, 50
- cnfEntryModeShift, 50
- cnfFontType, 51
- cnfI2C_addr, 51
- cnfLineNumber, 51
- command, 38
- ConfigureBacklight, 38
- ConfigureCursorBlink, 39
- ConfigureCursorOnOff, 40
- ConfigureDisplayOnOff, 40
- ConfigureEntryModeDir, 41
- ConfigureEntryModeShift, 42
- ConfigureFontType, 42
- ConfigureI2CAddr, 43
- ConfigureLineNumber, 44
- ddram_addr, 51
- GetDDRAMAddress, 44
- GetLineNumberCnf, 45
- InitializeScreen, 45

LCD, 37

SetDDRAMAddress, 46

write, 47

write4bits, 48

WriteInRam, 49

LED_PORT

- keepAliveLed.h, 137

LL_shift_data_ptr

- DisplayInterface, 29

LLElementCompare

- DisplayInterface, 27
- scheduler, 62

last_sum_value

- CpuLoad, 8

launchPeriodicTasks

- scheduler, 61

line

- T_Display_shift_data, 70

lineEmptyTab
DisplayInterface, 29

lineNumber_cnf
T_LCD_conf_struct, 73

LinkedList, 52

- AttachNewElement, 53
- curElement_ptr, 57
- firstElement, 58
- getCurrentElement, 54
- IsLLEmpty, 54
- LinkedList, 53
- MoveToNextElement, 55
- RemoveElement, 55
- ResetElementPtr, 57
- T_LL_element, 53

LinkedList.cpp, 150

LinkedList.h, 150

- CompareFctPtr_t, 151

LinkedList::T_LL_element, 73

- data_ptr, 74
- nextElement, 74

MAX_WAIT_TIME_US
dht22.cpp, 117

main

- main.cpp, 154

main.cpp, 152

- ISR, 153
- main, 154

main.h, 155

max_load

- CpuLoad, 9

MoveToNextElement
LinkedList, 55

NB_OF_SAMPLES
CpuLoad.h, 112

nextElement

- LinkedList::T_LL_element, 74

operator delete

- operators.cpp, 157
- operators.h, 158

operator new

- operators.cpp, 157
- operators.h, 159

operators.cpp, 156

- operator delete, 157
- operator new, 157

operators.h, 157

- operator delete, 158
- operator new, 159

p_DisplayInterface
T_ASW_cnf_struct, 66

p_TempSensor
T_ASW_cnf_struct, 67

p_cpupload
T_BSW_cnf_struct, 68

p_dht22
 T_BSW_cnf_struct, 68

p_dio
 T_BSW_cnf_struct, 69

p_i2c
 T_BSW_cnf_struct, 69

p_keepAliveLed
 T_ASW_cnf_struct, 67

p_lcd
 DisplayInterface, 29
 T_BSW_cnf_struct, 69

p_scheduler
 scheduler.cpp, 160
 scheduler.h, 163

p_timer
 T_BSW_cnf_struct, 69

p_usart
 T_BSW_cnf_struct, 69

p_USARTDebug
 T_ASW_cnf_struct, 67

PERIOD_MS_TASK_DISPLAY_CPU_LOAD
 debug.h, 115

PERIOD_MS_TASK_DISPLAY_SENSORS
 debug.h, 115

PERIOD_MS_TASK_LED
 keepAliveLed.h, 137

PERIOD_MS_TASK_TEMP_SENSOR
 TempSensor.h, 165

PINA_PTR
 dio_reg_atm2560.h, 125

PINB_PTR
 dio_reg_atm2560.h, 126

PINC_PTR
 dio_reg_atm2560.h, 126

PIND_PTR
 dio_reg_atm2560.h, 126

PINx_addr_mem
 dio, 21

PINx_idx_mem
 dio, 21

PIT_BEFORE_INVALID
 TempSensor.cpp, 164

PORT_CNF_IN
 dio.h, 121

PORT_CNF_OUT
 dio.h, 122

PORT_A
 dio.h, 121

PORT_B
 dio.h, 121

PORT_C
 dio.h, 121

PORT_D
 dio.h, 122

PORTA_PTR
 dio_reg_atm2560.h, 126

PORTB_CNF_DDRB
 dio_port_cnf.h, 123

PORTB_CNF_PORTB
 dio_port_cnf.h, 123

PORTB_PTR
 dio_reg_atm2560.h, 126

PORTC_PTR
 dio_reg_atm2560.h, 127

PORTD_PTR
 dio_reg_atm2560.h, 127

PRESCALER_PERIODIC_TIMER
 scheduler.h, 162

period
 scheduler::Task_t, 75

pit_number
 scheduler, 65

ports_init
 dio, 21

prescaler
 timer, 86

RS_PIN
 LCD.h, 148

RW_PIN
 LCD.h, 148

read
 dht22, 11

read_humidity
 TempSensor, 81

read_temperature
 TempSensor, 81

readTempSensor_task
 TempSensor, 79

RemoveElement
 LinkedList, 55

removePeriodicTask
 scheduler, 63

ResetElementPtr
 LinkedList, 57

SLA_ACK
 I2C.h, 134

START
 I2C.h, 134

SW_PERIOD_MS
 scheduler.h, 162

sample_cnt
 CpuLoad, 9

sample_idx
 CpuLoad, 9

sample_mem
 CpuLoad, 9

scheduler, 58
 addPeriodicTask, 60
 getPitNumber, 61
 LLElementCompare, 62
 launchPeriodicTasks, 61
 pit_number, 65
 removePeriodicTask, 63
 scheduler, 60
 startScheduling, 64

Task_t, 59
TasksLL_ptr, 65
scheduler.cpp, 159
 p_scheduler, 160
scheduler.h, 161
 p_scheduler, 163
 PRESCALER_PERIODIC_TIMER, 162
 SW_PERIOD_MS, 162
 TIMER_CTC_VALUE, 162
TaskPtr_t, 163
scheduler::Task_t, 74
 period, 75
 TaskPtr, 75
sendBool
 UsartDebug, 96
sendData
 UsartDebug, 97
sendInteger
 UsartDebug, 98
setBaudRate
 uart, 88
setBitRate
 I2C, 31
SetDDRAMAddress
 LCD, 46
setTxAddress
 I2C, 32
setValidity
 TempSensor, 80
shiftLine_task
 DisplayInterface, 28
size
 T_Display_shift_data, 70
startScheduling
 scheduler, 64
startTimer1
 timer, 85
stopTimer1
 timer, 85
str_cur_ptr
 T_Display_shift_data, 71
str_debug_main_menu
 debug.cpp, 113
str_start_ptr
 T_Display_shift_data, 71
T_ASW_cnf_struct, 66
 p_DisplayInterface, 66
 p_TempSensor, 67
 p_keepAliveLed, 67
 p_usartDebug, 67
T_BSW_cnf_struct, 68
 p_cpupload, 68
 p_dht22, 68
 p_dio, 69
 p_i2c, 69
 p_lcd, 69
 p_timer, 69
 p_usart, 69
T_Display_shift_data, 70
 line, 70
 size, 70
 str_cur_ptr, 71
 str_start_ptr, 71
 temporization, 71
T_DisplayInterface_LineDisplayMode
 DisplayInterface.h, 130
T_LCD_command
 LCD.h, 149
T_LCD_conf_struct, 71
 backlight_en, 72
 cursor_en, 72
 cursorBlink_en, 72
 display_en, 72
 entryModeDir, 72
 entryModeShift, 72
 fontType_cnf, 73
 i2c_addr, 73
 lineNumber_cnf, 73
T_LCD_config_mode
 LCD.h, 149
T_LCD_ram_area
 LCD.h, 149
T_LL_element
 LinkedList, 53
TIMER_CTC_VALUE
 scheduler.h, 162
Task_t
 scheduler, 59
TaskPtr
 scheduler::Task_t, 75
TaskPtr_t
 scheduler.h, 163
TasksLL_ptr
 scheduler, 65
TempSensor, 75
 getHumPtr, 78
 getHumidity, 77
 getTemp, 78
 getTempPtr, 79
 read_humidity, 81
 read_temperature, 81
 readTempSensor_task, 79
 setValidity, 80
 TempSensor, 76
 updateLastValidValues, 81
 valid_hum, 82
 valid坑, 82
 valid_temp, 82
 validity, 82
 validity_last_read, 82
TempSensor.cpp, 163
 PIT_BEFORE_INVALID, 164
TempSensor.h, 164
 PERIOD_MS_TASK_TEMP_SENSOR, 165
temporization
 T_Display_shift_data, 71

timer, 83
 configureTimer1, 84
 getTimer1Value, 85
 prescaler, 86
 startTimer1, 85
 stopTimer1, 85
 timer, 83
timer.cpp, 166
timer.h, 167
tx_address
 I2C, 33

USART_BAUDRATE
 bsw.h, 108

updateLastValidValues
 TempSensor, 81

usart, 86
 BaudRate, 91
 setBaudRate, 88
 usart, 87
 usart_init, 88
 usart_read, 89
 usart_sendString, 89
 usart_transmit, 90

usart.cpp, 168
usart.h, 168
usart_init
 usart, 88
usart_read
 usart, 89
usart_sendString
 usart, 89
usart_transmit
 usart, 90

UsartDebug, 91
 activateDebugMode, 92
 debug_state, 99
 debugModeActive_F, 99
 DebugModeManagement, 93
 DisplayCPUUpload_task, 94
 DisplaySensors_task, 95
 isDebugModeActive, 96
 sendBool, 96
 sendData, 97
 sendInteger, 98
 UsartDebug, 92

valid_hum
 TempSensor, 82

valid_pit
 TempSensor, 82

valid_temp
 TempSensor, 82

validity
 TempSensor, 82

validity_last_read
 TempSensor, 82

write