

Arduino

1.0

Generated by Doxygen 1.8.14



# Contents

<b>1</b>	<b>Class Index</b>	<b>1</b>
1.1	Class List . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
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	debug_mgt_state_struct_t Struct Reference . . . . .	10
3.2.1	Detailed Description . . . . .	10
3.2.2	Member Data Documentation . . . . .	10
3.2.2.1	main_state . . . . .	10
3.2.2.2	wdg_state . . . . .	10
3.3	DebugInterface Class Reference . . . . .	11
3.3.1	Detailed Description . . . . .	12
3.3.2	Constructor & Destructor Documentation . . . . .	12
3.3.2.1	DebugInterface() . . . . .	12
3.3.3	Member Function Documentation . . . . .	12
3.3.3.1	ClearScreen() . . . . .	12
3.3.3.2	nextLine() . . . . .	13
3.3.3.3	read() . . . . .	13
3.3.3.4	sendBool() . . . . .	14
3.3.3.5	sendChar() . . . . .	14
3.3.3.6	sendInteger() . . . . .	15
3.3.3.7	sendString() [1/2] . . . . .	16
3.3.3.8	sendString() [2/2] . . . . .	17
3.3.4	Member Data Documentation . . . . .	17
3.3.4.1	uart_drv_ptr . . . . .	18
3.4	DebugManagement Class Reference . . . . .	18
3.4.1	Detailed Description . . . . .	19
3.4.2	Constructor & Destructor Documentation . . . . .	19
3.4.2.1	DebugManagement() . . . . .	20
3.4.3	Member Function Documentation . . . . .	20
3.4.3.1	DebugModeManagement() . . . . .	20
3.4.3.2	DisplayData() . . . . .	21
3.4.3.3	DisplayPeriodicData_task() . . . . .	22
3.4.3.4	exitDebugMenu() . . . . .	23
3.4.3.5	getIftPtr() . . . . .	24

3.4.3.6	getInfoStringPtr()	24
3.4.3.7	getMenuStringPtr()	24
3.4.3.8	MainMenuManagement()	25
3.4.3.9	setInfoStringPtr()	25
3.4.3.10	systemReset()	26
3.4.3.11	WatchdogMenuManagement()	26
3.4.4	Member Data Documentation	27
3.4.4.1	debug_ift_ptr	27
3.4.4.2	debug_state	27
3.4.4.3	info_string_ptr	28
3.4.4.4	menu_string_ptr	28
3.4.4.5	tempSensor_ptr	28
3.5	dht22 Class Reference	28
3.5.1	Detailed Description	29
3.5.2	Constructor & Destructor Documentation	29
3.5.2.1	dht22()	29
3.5.3	Member Function Documentation	30
3.5.3.1	initializeCommunication()	30
3.5.3.2	read()	31
3.5.4	Member Data Documentation	32
3.5.4.1	dht22_port	32
3.5.4.2	dio_ptr	33
3.6	dio Class Reference	33
3.6.1	Detailed Description	34
3.6.2	Constructor & Destructor Documentation	34
3.6.2.1	dio()	34
3.6.3	Member Function Documentation	34
3.6.3.1	dio_changePortPinCnf()	34
3.6.3.2	dio_getPort()	35
3.6.3.3	dio_getPort_fast()	36

3.6.3.4	dio_invertPort()	37
3.6.3.5	dio_memorizePINaddress()	37
3.6.3.6	dio_setPort()	38
3.6.3.7	getDDRxAddress()	39
3.6.3.8	getPINxAddress()	40
3.6.3.9	getPORTxAddress()	40
3.6.3.10	ports_init()	41
3.6.4	Member Data Documentation	41
3.6.4.1	PINx_addr_mem	41
3.6.4.2	PINx_idx_mem	42
3.7	DisplayInterface Class Reference	42
3.7.1	Detailed Description	43
3.7.2	Constructor & Destructor Documentation	44
3.7.2.1	DisplayInterface()	44
3.7.3	Member Function Documentation	44
3.7.3.1	ClearFullScreen()	44
3.7.3.2	ClearLine()	45
3.7.3.3	ClearStringInDataStruct()	46
3.7.3.4	DisplayFullLine()	47
3.7.3.5	FindFirstCharAddr()	48
3.7.3.6	getDisplayDataPtr()	48
3.7.3.7	IsLineEmpty()	49
3.7.3.8	RefreshLine()	49
3.7.3.9	setLineAlignment()	50
3.7.3.10	setLineAlignmentAndRefresh()	51
3.7.3.11	shiftLine_task()	51
3.7.3.12	updateLineAndRefresh()	52
3.7.4	Member Data Documentation	53
3.7.4.1	display_data	53
3.7.4.2	dummy	53

3.7.4.3	isShiftInProgress . . . . .	53
3.7.4.4	p_lcd . . . . .	54
3.8	DisplayManagement Class Reference . . . . .	54
3.8.1	Detailed Description . . . . .	55
3.8.2	Constructor & Destructor Documentation . . . . .	55
3.8.2.1	DisplayManagement() . . . . .	55
3.8.3	Member Function Documentation . . . . .	56
3.8.3.1	DisplaySensorData_Task() . . . . .	56
3.8.3.2	GetIftPointer() . . . . .	57
3.8.3.3	GetTempSensorPtr() . . . . .	58
3.8.3.4	RemoveWelcomeMessage_Task() . . . . .	58
3.8.4	Member Data Documentation . . . . .	59
3.8.4.1	p_display_ift . . . . .	59
3.8.4.2	p_tempSensor . . . . .	59
3.9	I2C Class Reference . . . . .	59
3.9.1	Detailed Description . . . . .	60
3.9.2	Constructor & Destructor Documentation . . . . .	60
3.9.2.1	I2C() . . . . .	60
3.9.3	Member Function Documentation . . . . .	61
3.9.3.1	initializeBus() . . . . .	61
3.9.3.2	setBitRate() . . . . .	61
3.9.3.3	setTxAddress() . . . . .	62
3.9.3.4	writeByte() . . . . .	62
3.9.4	Member Data Documentation . . . . .	63
3.9.4.1	bitrate . . . . .	63
3.9.4.2	tx_address . . . . .	63
3.10	keepAliveLed Class Reference . . . . .	63
3.10.1	Detailed Description . . . . .	64
3.10.2	Constructor & Destructor Documentation . . . . .	64
3.10.2.1	keepAliveLed() . . . . .	64

---

3.10.3 Member Function Documentation . . . . .	64
3.10.3.1 blinkLed_task() . . . . .	65
3.11 LCD Class Reference . . . . .	65
3.11.1 Detailed Description . . . . .	67
3.11.2 Constructor & Destructor Documentation . . . . .	67
3.11.2.1 LCD() . . . . .	67
3.11.3 Member Function Documentation . . . . .	68
3.11.3.1 command() . . . . .	68
3.11.3.2 ConfigureBacklight() . . . . .	69
3.11.3.3 ConfigureCursorBlink() . . . . .	70
3.11.3.4 ConfigureCursorOnOff() . . . . .	70
3.11.3.5 ConfigureDisplayOnOff() . . . . .	71
3.11.3.6 ConfigureEntryModeDir() . . . . .	71
3.11.3.7 ConfigureEntryModeShift() . . . . .	72
3.11.3.8 ConfigureFontType() . . . . .	73
3.11.3.9 ConfigureI2CAddr() . . . . .	74
3.11.3.10 ConfigureLineNumber() . . . . .	74
3.11.3.11 GetDDRAMAddress() . . . . .	75
3.11.3.12 GetLineNumberCnf() . . . . .	75
3.11.3.13 InitializeScreen() . . . . .	76
3.11.3.14 SetDDRAMAddress() . . . . .	76
3.11.3.15 write() . . . . .	77
3.11.3.16 write4bits() . . . . .	78
3.11.3.17 WriteInRam() . . . . .	79
3.11.4 Member Data Documentation . . . . .	79
3.11.4.1 backlight_enable . . . . .	80
3.11.4.2 cnfCursorBlink . . . . .	80
3.11.4.3 cnfCursorOnOff . . . . .	80
3.11.4.4 cnfDisplayOnOff . . . . .	80
3.11.4.5 cnfEntryModeDir . . . . .	80

3.11.4.6 cnfEntryModeShift . . . . .	81
3.11.4.7 cnfFontType . . . . .	81
3.11.4.8 cnfl2C_addr . . . . .	81
3.11.4.9 cnfLineNumber . . . . .	81
3.11.4.10 ddram_addr . . . . .	81
3.11.4.11 i2c_drv_ptr . . . . .	82
3.12 LinkedList Class Reference . . . . .	82
3.12.1 Detailed Description . . . . .	83
3.12.2 Member Typedef Documentation . . . . .	83
3.12.2.1 T_LL_element . . . . .	84
3.12.3 Constructor & Destructor Documentation . . . . .	84
3.12.3.1 LinkedList() . . . . .	84
3.12.3.2 ~LinkedList() . . . . .	84
3.12.4 Member Function Documentation . . . . .	85
3.12.4.1 AttachNewElement() . . . . .	85
3.12.4.2 FindElement() . . . . .	85
3.12.4.3 getCurrentElement() . . . . .	86
3.12.4.4 IsLLEmpty() . . . . .	87
3.12.4.5 MoveToNextElement() . . . . .	87
3.12.4.6 RemoveElement() . . . . .	88
3.12.4.7 ResetElementPtr() . . . . .	88
3.12.5 Member Data Documentation . . . . .	89
3.12.5.1 curElement_ptr . . . . .	89
3.12.5.2 firstElement . . . . .	89
3.13 scheduler Class Reference . . . . .	89
3.13.1 Detailed Description . . . . .	90
3.13.2 Member Typedef Documentation . . . . .	91
3.13.2.1 Task_t . . . . .	91
3.13.3 Constructor & Destructor Documentation . . . . .	91
3.13.3.1 scheduler() . . . . .	91

---

3.13.4 Member Function Documentation . . . . .	91
3.13.4.1 addPeriodicTask() . . . . .	91
3.13.4.2 getPitNumber() . . . . .	92
3.13.4.3 getTaskCount() . . . . .	93
3.13.4.4 launchPeriodicTasks() . . . . .	93
3.13.4.5 LLElementCompare() . . . . .	94
3.13.4.6 removePeriodicTask() . . . . .	94
3.13.4.7 startScheduling() . . . . .	95
3.13.4.8 updateTaskPeriod() . . . . .	96
3.13.5 Member Data Documentation . . . . .	97
3.13.5.1 pit_number . . . . .	97
3.13.5.2 task_count . . . . .	97
3.13.5.3 TasksLL_ptr . . . . .	97
3.14 String Class Reference . . . . .	98
3.14.1 Detailed Description . . . . .	98
3.14.2 Constructor & Destructor Documentation . . . . .	99
3.14.2.1 String() [1/2] . . . . .	99
3.14.2.2 String() [2/2] . . . . .	99
3.14.2.3 ~String() . . . . .	100
3.14.3 Member Function Documentation . . . . .	100
3.14.3.1 appendBool() . . . . .	100
3.14.3.2 appendChar() . . . . .	101
3.14.3.3 appendInteger() . . . . .	102
3.14.3.4 appendString() . . . . .	103
3.14.3.5 Clear() . . . . .	103
3.14.3.6 ComputeStringSize() . . . . .	104
3.14.3.7 getSize() . . . . .	105
3.14.3.8 getString() . . . . .	105
3.14.4 Member Data Documentation . . . . .	106
3.14.4.1 size . . . . .	106

3.14.4.2 string . . . . .	106
3.15 T_ASW_init_cnf Struct Reference . . . . .	106
3.15.1 Detailed Description . . . . .	106
3.15.2 Member Data Documentation . . . . .	107
3.15.2.1 isDebugActivated . . . . .	107
3.15.2.2 isDisplayActivated . . . . .	107
3.15.2.3 isLEDActivated . . . . .	107
3.15.2.4 isTempSensorActivated . . . . .	107
3.16 T_display_data Struct Reference . . . . .	108
3.16.1 Detailed Description . . . . .	108
3.16.2 Member Data Documentation . . . . .	108
3.16.2.1 alignment . . . . .	109
3.16.2.2 display_str . . . . .	109
3.16.2.3 isEmpty . . . . .	109
3.16.2.4 mode . . . . .	109
3.16.2.5 shift_data . . . . .	109
3.17 T_Display_shift_data Struct Reference . . . . .	110
3.17.1 Detailed Description . . . . .	110
3.17.2 Member Data Documentation . . . . .	110
3.17.2.1 str_cur_ptr . . . . .	110
3.17.2.2 str_ptr . . . . .	111
3.17.2.3 temporization . . . . .	111
3.18 T_LCD_conf_struct Struct Reference . . . . .	111
3.18.1 Detailed Description . . . . .	111
3.18.2 Member Data Documentation . . . . .	112
3.18.2.1 backlight_en . . . . .	112
3.18.2.2 cursor_en . . . . .	112
3.18.2.3 cursorBlink_en . . . . .	112
3.18.2.4 display_en . . . . .	112
3.18.2.5 entryModeDir . . . . .	112

3.18.2.6	entryModeShift . . . . .	113
3.18.2.7	fontType_cnf . . . . .	113
3.18.2.8	i2c_addr . . . . .	113
3.18.2.9	i2c_bitrate . . . . .	113
3.18.2.10	lineNumber_cnf . . . . .	113
3.19	LinkedList::T_LL_element Struct Reference . . . . .	114
3.19.1	Detailed Description . . . . .	114
3.19.2	Member Data Documentation . . . . .	114
3.19.2.1	data_ptr . . . . .	114
3.19.2.2	nextElement . . . . .	114
3.20	scheduler::Task_t Struct Reference . . . . .	115
3.20.1	Detailed Description . . . . .	115
3.20.2	Member Data Documentation . . . . .	115
3.20.2.1	period . . . . .	115
3.20.2.2	TaskPtr . . . . .	115
3.21	TempSensor Class Reference . . . . .	115
3.21.1	Detailed Description . . . . .	116
3.21.2	Constructor & Destructor Documentation . . . . .	117
3.21.2.1	TempSensor() . . . . .	117
3.21.3	Member Function Documentation . . . . .	117
3.21.3.1	GetHumDecimal() . . . . .	117
3.21.3.2	getHumidity() . . . . .	118
3.21.3.3	GetHumInteger() . . . . .	118
3.21.3.4	getHumPtr() . . . . .	119
3.21.3.5	getTaskPeriod() . . . . .	119
3.21.3.6	getTemp() . . . . .	119
3.21.3.7	GetTempDecimal() . . . . .	120
3.21.3.8	GetTemplInteger() . . . . .	120
3.21.3.9	getTempPtr() . . . . .	121
3.21.3.10	GetValidity() . . . . .	121

3.21.3.11 <code>readTempSensor_task()</code>	122
3.21.3.12 <code>setValidity()</code>	122
3.21.3.13 <code>updateLastValidValues()</code>	123
3.21.3.14 <code>updateTaskPeriod()</code>	123
3.21.4 Member Data Documentation	124
3.21.4.1 <code>read_humidity</code>	124
3.21.4.2 <code>read_temperature</code>	125
3.21.4.3 <code>task_period</code>	125
3.21.4.4 <code>valid_hum</code>	125
3.21.4.5 <code>valid_pit</code>	125
3.21.4.6 <code>valid_temp</code>	125
3.21.4.7 <code>validity</code>	126
3.21.4.8 <code>validity_last_read</code>	126
3.22 timer Class Reference	126
3.22.1 Detailed Description	127
3.22.2 Constructor & Destructor Documentation	127
3.22.2.1 <code>timer()</code>	127
3.22.3 Member Function Documentation	127
3.22.3.1 <code>configureTimer1()</code>	127
3.22.3.2 <code>getTimer1Value()</code>	128
3.22.3.3 <code>startTimer1()</code>	129
3.22.3.4 <code>stopTimer1()</code>	129
3.22.4 Member Data Documentation	129
3.22.4.1 <code>prescaler</code>	129
3.23 usart Class Reference	130
3.23.1 Detailed Description	130
3.23.2 Constructor & Destructor Documentation	130
3.23.2.1 <code>usart()</code>	130
3.23.3 Member Function Documentation	131
3.23.3.1 <code>setBaudRate()</code>	131

3.23.3.2  uart_init()	132
3.23.3.3  uart_read()	132
3.23.3.4  uart_sendByte()	132
3.23.3.5  uart_sendString()	133
3.23.3.6  uart_transmit()	134
3.23.4  Member Data Documentation	135
3.23.4.1  BaudRate	135
3.24  Watchdog Class Reference	135
3.24.1  Detailed Description	136
3.24.2  Constructor & Destructor Documentation	136
3.24.2.1  Watchdog() [1/2]	136
3.24.2.2  Watchdog() [2/2]	136
3.24.3  Member Function Documentation	137
3.24.3.1  disable()	137
3.24.3.2  enable()	137
3.24.3.3  reset()	138
3.24.3.4  SystemReset()	139
3.24.3.5  timeoutUpdate()	139
<b>4  File Documentation</b>	<b>141</b>
4.1  asw.cpp File Reference	141
4.1.1  Detailed Description	142
4.1.2  Function Documentation	142
4.1.2.1  asw_init()	142
4.2  asw.h File Reference	143
4.2.1  Detailed Description	143
4.2.2  Function Documentation	143
4.2.2.1  asw_init()	144
4.3  bsw.cpp File Reference	144
4.3.1  Detailed Description	145
4.3.2  Function Documentation	145

4.3.2.1	<code>bsw_init()</code>	145
4.4	<code>bsw.h</code> File Reference	146
4.4.1	Detailed Description	146
4.4.2	Function Documentation	146
4.4.2.1	<code>bsw_init()</code>	147
4.5	<code>CpuLoad.cpp</code> File Reference	147
4.5.1	Detailed Description	148
4.5.2	Variable Documentation	148
4.5.2.1	<code>p_global_BSW_cpupload</code>	148
4.6	<code>CpuLoad.h</code> File Reference	148
4.6.1	Detailed Description	149
4.6.2	Macro Definition Documentation	149
4.6.2.1	<code>NB_OF_SAMPLES</code>	149
4.6.3	Variable Documentation	149
4.6.3.1	<code>p_global_BSW_cpupload</code>	149
4.7	<code>DebugInterface.cpp</code> File Reference	150
4.7.1	Detailed Description	150
4.7.2	Variable Documentation	150
4.7.2.1	<code>p_global_ASW_DebugInterface</code>	150
4.8	<code>DebugInterface.h</code> File Reference	151
4.8.1	Detailed Description	151
4.8.2	Macro Definition Documentation	151
4.8.2.1	<code>USART_BAUDRATE</code>	152
4.8.3	Variable Documentation	152
4.8.3.1	<code>p_global_ASW_DebugInterface</code>	152
4.9	<code>DebugManagement.cpp</code> File Reference	152
4.9.1	Detailed Description	153
4.9.2	Variable Documentation	153
4.9.2.1	<code>p_global_ASW_DebugManagement</code>	153
4.9.2.2	<code>str_debug_info_message_empty</code>	153

4.9.2.3	str_debug_info_message_wdg_tmo_updated . . . . .	154
4.9.2.4	str_debug_info_message_wrong_menu_selection . . . . .	154
4.9.2.5	str_debug_main_menu . . . . .	154
4.9.2.6	str_debug_wdg_menu . . . . .	154
4.9.2.7	str_debug_wdg_timeout_update_selection . . . . .	155
4.10	DebugManagement.h File Reference . . . . .	155
4.10.1	Detailed Description . . . . .	156
4.10.2	Macro Definition Documentation . . . . .	156
4.10.2.1	PERIOD_MS_TASK_DISPLAY_CPU_LOAD . . . . .	156
4.10.2.2	PERIOD_MS_TASK_DISPLAY_DEBUG_DATA . . . . .	156
4.10.3	Enumeration Type Documentation . . . . .	156
4.10.3.1	debug_mgt_main_menu_state_t . . . . .	156
4.10.3.2	debug_mgt_wdg_state_t . . . . .	157
4.10.4	Variable Documentation . . . . .	157
4.10.4.1	p_global_ASW_DebugManagement . . . . .	157
4.11	dht22.cpp File Reference . . . . .	157
4.11.1	Detailed Description . . . . .	158
4.11.2	Macro Definition Documentation . . . . .	158
4.11.2.1	MAX_WAIT_TIME_US . . . . .	158
4.11.3	Variable Documentation . . . . .	159
4.11.3.1	p_global_BSW_dht22 . . . . .	159
4.12	dht22.h File Reference . . . . .	159
4.12.1	Detailed Description . . . . .	159
4.12.2	Variable Documentation . . . . .	160
4.12.2.1	p_global_BSW_dht22 . . . . .	160
4.13	dio.cpp File Reference . . . . .	160
4.13.1	Detailed Description . . . . .	160
4.13.2	Variable Documentation . . . . .	161
4.13.2.1	p_global_BSW_dio . . . . .	161
4.14	dio.h File Reference . . . . .	161

4.14.1	Detailed Description	162
4.14.2	Macro Definition Documentation	162
4.14.2.1	DECODE_PIN	162
4.14.2.2	DECODE_PORT	162
4.14.2.3	ENCODE_PORT	163
4.14.2.4	PORT_CNF_IN	163
4.14.2.5	PORT_CNF_OUT	163
4.14.3	Variable Documentation	163
4.14.3.1	p_global_BSW_dio	163
4.15	dio_port_cnf.h File Reference	164
4.15.1	Detailed Description	164
4.15.2	Macro Definition Documentation	164
4.15.2.1	PORT_A	164
4.15.2.2	PORT_B	165
4.15.2.3	PORT_C	165
4.15.2.4	PORT_D	165
4.15.2.5	PORTB_CNF_DDRB	165
4.15.2.6	PORTB_CNF_PORTB	166
4.16	dio_reg_atm2560.h File Reference	166
4.16.1	Detailed Description	167
4.16.2	Macro Definition Documentation	167
4.16.2.1	DDRA_PTR	167
4.16.2.2	DDR_B_PTR	167
4.16.2.3	DDRC_PTR	167
4.16.2.4	DDR_D_PTR	168
4.16.2.5	PINA_PTR	168
4.16.2.6	PINB_PTR	168
4.16.2.7	PINC_PTR	168
4.16.2.8	PIND_PTR	168
4.16.2.9	PORTA_PTR	169

4.16.2.10 PORTB_PTR . . . . .	169
4.16.2.11 PORTC_PTR . . . . .	169
4.16.2.12 PORTD_PTR . . . . .	169
4.17 DisplayInterface.cpp File Reference . . . . .	169
4.17.1 Detailed Description . . . . .	170
4.17.2 Variable Documentation . . . . .	170
4.17.2.1 p_global_ASW_DisplayInterface . . . . .	170
4.18 DisplayInterface.h File Reference . . . . .	170
4.18.1 Detailed Description . . . . .	171
4.18.2 Macro Definition Documentation . . . . .	171
4.18.2.1 DISPLAY_LINE_SHIFT_PERIOD_MS . . . . .	171
4.18.2.2 DISPLAY_LINE_SHIFT_TEMPO_TIME . . . . .	172
4.18.3 Enumeration Type Documentation . . . . .	172
4.18.3.1 T_DisplayInterface_LineAlignment . . . . .	172
4.18.3.2 T_DisplayInterface_LineDisplayMode . . . . .	172
4.18.4 Variable Documentation . . . . .	173
4.18.4.1 p_global_ASW_DisplayInterface . . . . .	173
4.19 DisplayManagement.cpp File Reference . . . . .	173
4.19.1 Detailed Description . . . . .	173
4.19.2 Variable Documentation . . . . .	174
4.19.2.1 p_global_ASW_DisplayManagement . . . . .	174
4.20 DisplayManagement.h File Reference . . . . .	174
4.20.1 Detailed Description . . . . .	175
4.20.2 Macro Definition Documentation . . . . .	175
4.20.2.1 DISPLAY_MGT_I2C_BITRATE . . . . .	175
4.20.2.2 DISPLAY_MGT_LCD_I2C_ADDR . . . . .	175
4.20.2.3 DISPLAY_MGT_LINE_HUM . . . . .	176
4.20.2.4 DISPLAY_MGT_LINE_TEMP . . . . .	176
4.20.2.5 DISPLAY_MGT_PERIOD_TASK_SENSOR . . . . .	176
4.20.2.6 DISPLAY_MGT_PERIOD_WELCOME_MSG_REMOVAL . . . . .	176

4.20.3 Variable Documentation . . . . .	176
4.20.3.1 humidityDisplayString . . . . .	176
4.20.3.2 LCD_init_cnf . . . . .	177
4.20.3.3 noSensorDisplayString . . . . .	177
4.20.3.4 p_global_ASW_DisplayManagement . . . . .	177
4.20.3.5 tempDisplayString . . . . .	177
4.20.3.6 welcomeMessageString . . . . .	178
4.21 I2C.cpp File Reference . . . . .	178
4.21.1 Detailed Description . . . . .	178
4.21.2 Variable Documentation . . . . .	178
4.21.2.1 p_global_BSW_i2c . . . . .	179
4.22 I2C.h File Reference . . . . .	179
4.22.1 Detailed Description . . . . .	179
4.22.2 Macro Definition Documentation . . . . .	180
4.22.2.1 DATA_ACK . . . . .	180
4.22.2.2 SLA_ACK . . . . .	180
4.22.2.3 START . . . . .	180
4.22.3 Variable Documentation . . . . .	180
4.22.3.1 p_global_BSW_i2c . . . . .	180
4.23 int.cpp File Reference . . . . .	181
4.23.1 Detailed Description . . . . .	181
4.23.2 Function Documentation . . . . .	181
4.23.2.1 ISR() [1/2] . . . . .	182
4.23.2.2 ISR() [2/2] . . . . .	182
4.24 keepAliveLed.cpp File Reference . . . . .	183
4.24.1 Detailed Description . . . . .	183
4.24.2 Variable Documentation . . . . .	184
4.24.2.1 p_global_ASW_keepAliveLed . . . . .	184
4.25 keepAliveLed.h File Reference . . . . .	184
4.25.1 Detailed Description . . . . .	185

---

4.25.2 Macro Definition Documentation . . . . .	185
4.25.2.1 LED_PORT . . . . .	185
4.25.2.2 PERIOD_MS_TASK_LED . . . . .	185
4.25.3 Variable Documentation . . . . .	185
4.25.3.1 p_global_ASW_keepAliveLed . . . . .	185
4.26 LCD.cpp File Reference . . . . .	186
4.26.1 Detailed Description . . . . .	186
4.26.2 Variable Documentation . . . . .	186
4.26.2.1 p_global_BSW_lcd . . . . .	186
4.27 LCD.h File Reference . . . . .	187
4.27.1 Detailed Description . . . . .	188
4.27.2 Macro Definition Documentation . . . . .	188
4.27.2.1 BACKLIGHT_PIN . . . . .	189
4.27.2.2 EN_PIN . . . . .	189
4.27.2.3 LCD_CNF_BACKLIGHT_OFF . . . . .	189
4.27.2.4 LCD_CNF_BACKLIGHT_ON . . . . .	189
4.27.2.5 LCD_CNF_CURSOR_BLINK_OFF . . . . .	189
4.27.2.6 LCD_CNF_CURSOR_BLINK_ON . . . . .	190
4.27.2.7 LCD_CNF_CURSOR_OFF . . . . .	190
4.27.2.8 LCD_CNF_CURSOR_ON . . . . .	190
4.27.2.9 LCD_CNF_DISPLAY_OFF . . . . .	190
4.27.2.10 LCD_CNF_DISPLAY_ON . . . . .	190
4.27.2.11 LCD_CNF_ENTRY_MODE_DIRECTION_LEFT . . . . .	191
4.27.2.12 LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT . . . . .	191
4.27.2.13 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF . . . . .	191
4.27.2.14 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON . . . . .	191
4.27.2.15 LCD_CNF_FONT_5_11 . . . . .	191
4.27.2.16 LCD_CNF_FONT_5_8 . . . . .	192
4.27.2.17 LCD_CNF_ONE_LINE . . . . .	192
4.27.2.18 LCD_CNF_SHIFT_ID . . . . .	192

4.27.2.19 LCD_CNF_SHIFT_SH . . . . .	192
4.27.2.20 LCD_CNF_TWO_LINE . . . . .	192
4.27.2.21 LCD_DISPLAY_CTRL_FIELD_B . . . . .	193
4.27.2.22 LCD_DISPLAY_CTRL_FIELD_C . . . . .	193
4.27.2.23 LCD_DISPLAY_CTRL_FIELD_D . . . . .	193
4.27.2.24 LCD_FCT_SET_FIELD_DL . . . . .	193
4.27.2.25 LCD_FCT_SET_FIELD_F . . . . .	193
4.27.2.26 LCD_FCT_SET_FIELD_N . . . . .	194
4.27.2.27 LCD_INST_CLR_DISPLAY_BIT . . . . .	194
4.27.2.28 LCD_INST_DISPLAY_CTRL . . . . .	194
4.27.2.29 LCD_INST_ENTRY_MODE_SET . . . . .	194
4.27.2.30 LCD_INST_FUNCTION_SET . . . . .	194
4.27.2.31 LCD_INST_SET_DDRAM_ADDR . . . . .	195
4.27.2.32 LCD_RAM_1_LINE_MAX . . . . .	195
4.27.2.33 LCD_RAM_1_LINE_MIN . . . . .	195
4.27.2.34 LCD_RAM_2_LINES_MAX_1 . . . . .	195
4.27.2.35 LCD_RAM_2_LINES_MAX_2 . . . . .	195
4.27.2.36 LCD_RAM_2_LINES_MIN_1 . . . . .	196
4.27.2.37 LCD_RAM_2_LINES_MIN_2 . . . . .	196
4.27.2.38 LCD_SIZE_NB_CHAR_PER_LINE . . . . .	196
4.27.2.39 LCD_SIZE_NB_LINES . . . . .	196
4.27.2.40 LCD_WAIT_CLR_RETURN . . . . .	196
4.27.2.41 LCD_WAIT_OTHER_MODES . . . . .	197
4.27.2.42 RS_PIN . . . . .	197
4.27.2.43 RW_PIN . . . . .	197
4.27.3 Enumeration Type Documentation . . . . .	197
4.27.3.1 T_LCD_command . . . . .	197
4.27.3.2 T_LCD_config_mode . . . . .	198
4.27.3.3 T_LCD_ram_area . . . . .	198
4.27.4 Variable Documentation . . . . .	198

4.27.4.1	p_global_BSW_lcd . . . . .	198
4.28	LinkedList.cpp File Reference . . . . .	199
4.28.1	Detailed Description . . . . .	199
4.29	LinkedList.h File Reference . . . . .	199
4.29.1	Detailed Description . . . . .	200
4.29.2	Typedef Documentation . . . . .	200
4.29.2.1	CompareFctPtr_t . . . . .	200
4.30	main.cpp File Reference . . . . .	200
4.30.1	Detailed Description . . . . .	201
4.30.2	Macro Definition Documentation . . . . .	201
4.30.2.1	DEBUG_ACTIVE_PORT . . . . .	201
4.30.3	Function Documentation . . . . .	201
4.30.3.1	main() . . . . .	202
4.30.4	Variable Documentation . . . . .	202
4.30.4.1	ASW_init_cnf . . . . .	202
4.30.4.2	isDebugModeActivated . . . . .	202
4.31	main.h File Reference . . . . .	203
4.31.1	Detailed Description . . . . .	203
4.31.2	Variable Documentation . . . . .	203
4.31.2.1	ASW_init_cnf . . . . .	203
4.31.2.2	isDebugModeActivated . . . . .	204
4.32	operators.cpp File Reference . . . . .	204
4.32.1	Detailed Description . . . . .	204
4.32.2	Function Documentation . . . . .	205
4.32.2.1	operator delete() . . . . .	205
4.32.2.2	operator new() . . . . .	205
4.33	operators.h File Reference . . . . .	205
4.33.1	Detailed Description . . . . .	206
4.33.2	Function Documentation . . . . .	206
4.33.2.1	operator delete() . . . . .	206

4.33.2.2 operator new() . . . . .	207
4.34 scheduler.cpp File Reference . . . . .	207
4.34.1 Detailed Description . . . . .	208
4.34.2 Variable Documentation . . . . .	208
4.34.2.1 p_global_scheduler . . . . .	208
4.35 scheduler.h File Reference . . . . .	208
4.35.1 Detailed Description . . . . .	209
4.35.2 Macro Definition Documentation . . . . .	209
4.35.2.1 PRESCALER_PERIODIC_TIMER . . . . .	209
4.35.2.2 SW_PERIOD_MS . . . . .	209
4.35.2.3 TIMER_CTC_VALUE . . . . .	210
4.35.3 Typedef Documentation . . . . .	210
4.35.3.1 TaskPtr_t . . . . .	210
4.35.4 Variable Documentation . . . . .	210
4.35.4.1 p_global_scheduler . . . . .	210
4.36 String.cpp File Reference . . . . .	210
4.36.1 Detailed Description . . . . .	211
4.37 String.h File Reference . . . . .	211
4.37.1 Detailed Description . . . . .	211
4.38 TempSensor.cpp File Reference . . . . .	212
4.38.1 Detailed Description . . . . .	212
4.38.2 Macro Definition Documentation . . . . .	212
4.38.2.1 PIT_BEFORE_INVALID . . . . .	213
4.38.3 Variable Documentation . . . . .	213
4.38.3.1 p_global_ASW_TempSensor . . . . .	213
4.39 TempSensor.h File Reference . . . . .	213
4.39.1 Detailed Description . . . . .	214
4.39.2 Macro Definition Documentation . . . . .	214
4.39.2.1 DHT22_PORT . . . . .	214
4.39.2.2 PERIOD_MS_TASK_TEMP_SENSOR . . . . .	214

4.39.3 Variable Documentation . . . . .	214
4.39.3.1 p_global_ASW_TempSensor . . . . .	214
4.40 timer.cpp File Reference . . . . .	215
4.40.1 Detailed Description . . . . .	215
4.40.2 Variable Documentation . . . . .	215
4.40.2.1 p_global_BSW_timer . . . . .	215
4.41 timer.h File Reference . . . . .	216
4.41.1 Detailed Description . . . . .	216
4.41.2 Variable Documentation . . . . .	216
4.41.2.1 p_global_BSW_timer . . . . .	216
4.42 usart.cpp File Reference . . . . .	217
4.42.1 Detailed Description . . . . .	217
4.42.2 Variable Documentation . . . . .	217
4.42.2.1 p_global_BSW_usart . . . . .	217
4.43 usart.h File Reference . . . . .	218
4.43.1 Detailed Description . . . . .	218
4.43.2 Variable Documentation . . . . .	218
4.43.2.1 p_global_BSW_usart . . . . .	218
4.44 Watchdog.cpp File Reference . . . . .	219
4.44.1 Detailed Description . . . . .	219
4.44.2 Macro Definition Documentation . . . . .	219
4.44.2.1 WDG_TIMEOUT_DEFAULT_MS . . . . .	220
4.44.3 Variable Documentation . . . . .	220
4.44.3.1 p_global_BSW_wdg . . . . .	220
4.45 Watchdog.h File Reference . . . . .	220
4.45.1 Detailed Description . . . . .	221
4.45.2 Macro Definition Documentation . . . . .	221
4.45.2.1 WDG_TMO_120MS . . . . .	221
4.45.2.2 WDG_TMO_15MS . . . . .	222
4.45.2.3 WDG_TMO_1S . . . . .	222
4.45.2.4 WDG_TMO_250MS . . . . .	222
4.45.2.5 WDG_TMO_2S . . . . .	222
4.45.2.6 WDG_TMO_30MS . . . . .	222
4.45.2.7 WDG_TMO_4S . . . . .	223
4.45.2.8 WDG_TMO_500MS . . . . .	223
4.45.2.9 WDG_TMO_60MS . . . . .	223
4.45.2.10 WDG_TMO_8S . . . . .	223
4.45.3 Variable Documentation . . . . .	223
4.45.3.1 p_global_BSW_wdg . . . . .	223
Index . . . . .	225

# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">CpuLoad</a>	Class defining CPU load libraries . . . . .	5
<a href="#">debug_mgt_state_struct_t</a>	Structure containing all debug states . . . . .	10
<a href="#">DebugInterface</a>	Class used for debugging on usart link . . . . .	11
<a href="#">DebugManagement</a>	Debug management class . . . . .	18
<a href="#">dht22</a>	DHT 22 driver class . . . . .	28
<a href="#">dio</a>	DIO class . . . . .	33
<a href="#">DisplayInterface</a>	Display interface services class . . . . .	42
<a href="#">DisplayManagement</a>	Display management class . . . . .	54
<a href="#">I2C</a>	Two-wire serial interface ( <a href="#">I2C</a> ) class definition . . . . .	59
<a href="#">keepAliveLed</a>	Class for keep-alive LED blinking . . . . .	63
<a href="#">LCD</a>	Class for <a href="#">LCD</a> S2004A display driver . . . . .	65
<a href="#">LinkedList</a>	Linked list class . . . . .	82
<a href="#">scheduler</a>	Scheduler class . . . . .	89
<a href="#">String</a>	<a href="#">String</a> management class . . . . .	98
<a href="#">T_ASW_init_cnf</a>	ASW initialization configuration structure . . . . .	106
<a href="#">T_display_data</a>	Structure containing display data . . . . .	108
<a href="#">T_Display_shift_data</a>	Structure containing shift data . . . . .	110
<a href="#">T_LCD_conf_struct</a>	Structure defining <a href="#">LCD</a> configuration . . . . .	111

<a href="#">LinkedList::T_LL_element</a>	
Type defining a linked list element	114
<a href="#">scheduler::Task_t</a>	
Type defining a task structure	115
<a href="#">TempSensor</a>	
Class for temperature sensor	115
<a href="#">timer</a>	
Class defining a timer	126
<a href="#">usart</a>	
USART serial bus class	130
<a href="#">Watchdog</a>	
<a href="#">Watchdog</a> management class	135

# Chapter 2

## File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">asw.cpp</a>	ASW main file . . . . .	141
<a href="#">asw.h</a>	ASW main header file . . . . .	143
<a href="#">bsw.cpp</a>	BSW main file . . . . .	144
<a href="#">bsw.h</a>	BSW main header file . . . . .	146
<a href="#">CpuLoad.cpp</a>	Defines functions of class <a href="#">CpuLoad</a> . . . . .	147
<a href="#">CpuLoad.h</a>	<a href="#">CpuLoad</a> class header file . . . . .	148
<a href="#">DebugInterface.cpp</a>	This file defines classes for log and debug data transmission on USART link . . . . .	150
<a href="#">DebugInterface.h</a>	Header file for debug and logging functions . . . . .	151
<a href="#">DebugManagement.cpp</a>	Debug management class source file . . . . .	152
<a href="#">DebugManagement.h</a>	Debug management class header file . . . . .	155
<a href="#">dht22.cpp</a>	This file defines classes for DHT22 driver . . . . .	157
<a href="#">dht22.h</a>	DHT22 driver header file . . . . .	159
<a href="#">dio.cpp</a>	DIO library . . . . .	160
<a href="#">dio.h</a>	DIO library header file . . . . .	161
<a href="#">dio_port_cnf.h</a>	Digital ports configuration file . . . . .	164
<a href="#">dio_reg_atm2560.h</a>	Defines DIO register addresses for ATMEGA2560 . . . . .	166
<a href="#">DisplayInterface.cpp</a>	Source code file for display services . . . . .	169
<a href="#">DisplayInterface.h</a>	<a href="#">DisplayInterface</a> class header file . . . . .	170

DisplayManagement.cpp	Display management source file	173
DisplayManagement.h	Display management class header file	174
I2C.cpp	Two-wire interface ( <a href="#">I2C</a> ) source file	178
I2C.h	<a href="#">I2C</a> class header file	179
int.cpp	Interrupt management source file	181
keepAliveLed.cpp	Definition of function for class <a href="#">keepAliveLed</a>	183
keepAliveLed.h	Class <a href="#">keepAliveLed</a> header file	184
LCD.cpp	<a href="#">LCD</a> class source file	186
LCD.h	<a href="#">LCD</a> class header file	187
LinkedList.cpp	Linked List library source file	199
LinkedList.h	Linked List library header file	199
main.cpp	Background task file	200
main.h	Background task header file	203
operators.cpp	C++ operators definitions	204
operators.h	C++ operators definitions header file	205
scheduler.cpp	Defines scheduler class	207
scheduler.h	Scheduler class header file	208
String.cpp	<a href="#">String</a> class source file	210
String.h	<a href="#">String</a> class header file	211
TempSensor.cpp	Defines function of class <a href="#">TempSensor</a>	212
TempSensor.h	Class <a href="#">TempSensor</a> header file	213
timer.cpp	Defines function for class timer	215
timer.h	Timer class header file	216
uart.cpp	BSW library for USART	217
uart.h	Header file for USART library	218
Watchdog.cpp	Class <a href="#">Watchdog</a> source code file	219
Watchdog.h	Class <a href="#">Watchdog</a> header file	220

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

##### Returns

Nothing

Definition at line 20 of file CpuLoad.cpp.

### 3.1.3 Member Function Documentation

#### 3.1.3.1 ComputeCPUload()

```
void CpuLoad::ComputeCPUload ( )
```

Computes current CPU load.

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

##### Returns

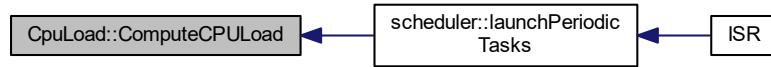
Nothing

Definition at line 40 of file CpuLoad.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.1.3.2 getAverageCPULoad()

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

Get average CPU load value.

This function returns the average CPU load value

#### Returns

Average CPU load value

Definition at line 58 of file CpuLoad.h.

Here is the caller graph for this function:



### 3.1.3.3 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 47 of file CpuLoad.h.

Here is the caller graph for this function:



### 3.1.3.4 getMaxCPUload()

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

Get maximum CPU load value.

This function returns the maximum CPU load value

#### Returns

Maximum CPU load value

Definition at line 69 of file CpuLoad.h.

Here is the caller graph for this function:



## 3.1.4 Member Data Documentation

### 3.1.4.1 avg\_load

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

Average CPU load based on the last 50 cycles

Definition at line 76 of file CpuLoad.h.

### 3.1.4.2 current\_load

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

Current CPU load (load of last cycle)

Definition at line 75 of file CpuLoad.h.

#### 3.1.4.3 last\_sum\_value

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

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

Definition at line 81 of file CpuLoad.h.

#### 3.1.4.4 max\_load

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

Maximum CPU load since power on

Definition at line 77 of file CpuLoad.h.

#### 3.1.4.5 sample\_cnt

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

Number of samples used to compute average load

Definition at line 78 of file CpuLoad.h.

#### 3.1.4.6 sample\_idx

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

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

Definition at line 80 of file CpuLoad.h.

#### 3.1.4.7 sample\_mem

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

Memorization of the last NB\_OF\_SAMPLES measures

Definition at line 79 of file CpuLoad.h.

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

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

## 3.2 debug\_mgt\_state\_struct\_t Struct Reference

Structure containing all debug states.

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

### Public Attributes

- `debug_mgt_main_menu_state_t main_state`
- `debug_mgt_wdg_state_t wdg_state`

#### 3.2.1 Detailed Description

Structure containing all debug states.

Definition at line 40 of file DebugManagement.h.

#### 3.2.2 Member Data Documentation

##### 3.2.2.1 main\_state

```
debug_mgt_main_menu_state_t debug_mgt_state_struct_t::main_state
```

Current main menu state

Definition at line 42 of file DebugManagement.h.

##### 3.2.2.2 wdg\_state

```
debug_mgt_wdg_state_t debug_mgt_state_struct_t::wdg_state
```

Current state of watchdog management

Definition at line 43 of file DebugManagement.h.

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

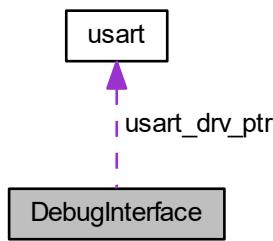
- [DebugManagement.h](#)

### 3.3 DebugInterface Class Reference

Class used for debugging on usart link.

```
#include <DebugInterface.h>
```

Collaboration diagram for DebugInterface:



#### Public Member Functions

- [DebugInterface \(\)](#)  
    *Class DebugInterface constructor.*
- [void sendInteger \(uint16\\_t data, uint8\\_t base\)](#)  
    *Send a integer data on USART link.*
- [void sendBool \(bool data, bool isText\)](#)  
    *Send a boolean data on USART link.*
- [void sendString \(String \\*str\)](#)  
    *Send a string on USART link.*
- [void sendString \(uint8\\_t \\*str\)](#)  
    *Send a chain of characters on USART link.*
- [void sendChar \(uint8\\_t chr\)](#)  
    *Send a single character on USART link.*
- [uint8\\_t read \(\)](#)  
    *USART read function.*
- [void nextLine \(\)](#)  
    *Go to next line function.*
- [void ClearScreen \(\)](#)  
    *Screen clearing function.*

#### Private Attributes

- [uart \\* usart\\_drv\\_ptr](#)

### 3.3.1 Detailed Description

Class used for debugging on usart link.

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

Definition at line 21 of file DebugInterface.h.

### 3.3.2 Constructor & Destructor Documentation

#### 3.3.2.1 DebugInterface()

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

Class [DebugInterface](#) constructor.

Initializes the class [DebugInterface](#). It creates a new instance of USART driver of needed.

**Returns**

Nothing

Definition at line 22 of file DebugInterface.cpp.

### 3.3.3 Member Function Documentation

#### 3.3.3.1 ClearScreen()

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

Screen clearing function.

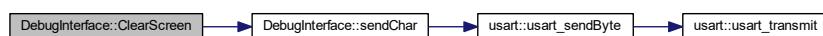
This function clears the entire display by sending the character on the USART line.

**Returns**

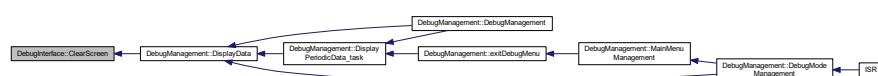
Nothing

Definition at line 77 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.3.3.2 nextLine()

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

Go to next line function.

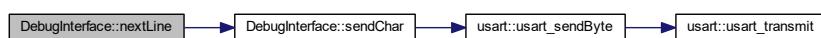
This function goes to the next line on the console display. It sends the two characters and on the USART line.

Returns

Nothing

Definition at line 71 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.3.3.3 read()

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

USART read function.

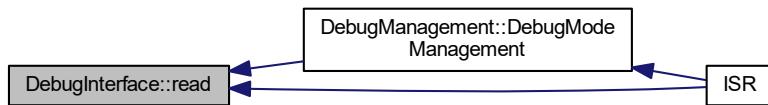
This function will read the last received byte on USART link

Returns

Received byte

Definition at line 82 of file DebugInterface.h.

Here is the caller graph for this function:



### 3.3.3.4 sendBool()

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

Send a boolean data on USART link.

This function sends the requested boolean on USART link by calling driver's transmission function. The boolean data is first converted into a string and then sent. The parameter `isText` defines if the data is converted into a string (true/false) or an integer (1/0).

#### Parameters

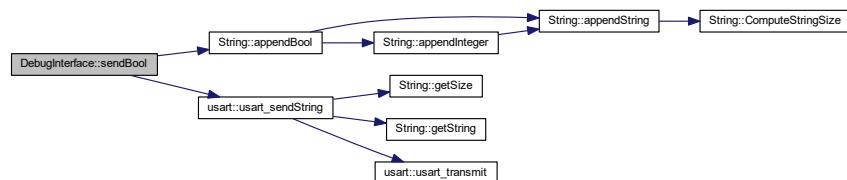
in	<code>data</code>	boolean data to be sent
in	<code>isText</code>	<a href="#">String</a> conversion configuration

#### Returns

Nothing

Definition at line 62 of file `DebugInterface.cpp`.

Here is the call graph for this function:



### 3.3.3.5 sendChar()

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

Send a single character on USART link.

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

#### Parameters

in	<code>chr</code>	Character to send.
----	------------------	--------------------

**Returns**

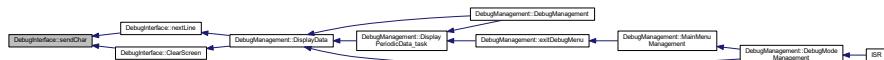
Nothing

Definition at line 44 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.3.3.6 sendInteger()**

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

Send a integer data on USART link.

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

**Parameters**

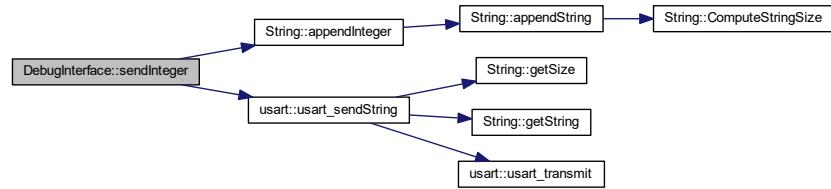
in	<b>data</b>	integer data to be sent
in	<b>base</b>	numerical base used to convert integer into string (between 2 and 36)

**Returns**

Nothing

Definition at line 49 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.3.3.7 sendString() [1/2]

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

Send a string on USART link.

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

#### Parameters

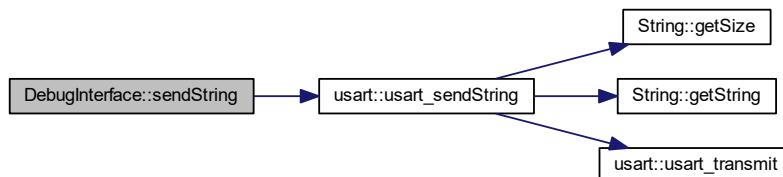
in	<i>str</i>	Pointer to the string being sent
----	------------	----------------------------------

#### Returns

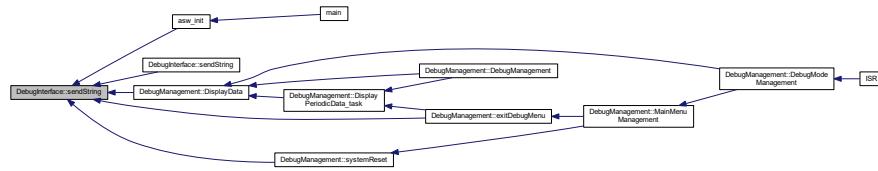
Nothing

Definition at line 31 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.3.3.8 sendString() [2/2]

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

Send a chain of characters on USART link.

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

#### Parameters

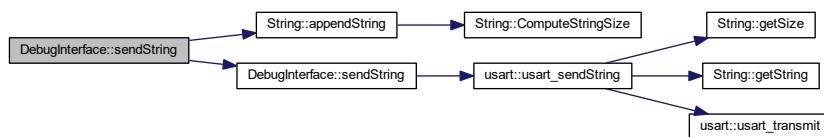
in	<i>str</i>	Pointer to the chain to send.
----	------------	-------------------------------

#### Returns

Nothing

Definition at line 37 of file DebugInterface.cpp.

Here is the call graph for this function:



## 3.3.4 Member Data Documentation

### 3.3.4.1 usart\_drv\_ptr

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

Pointer to USART driver object

Definition at line 107 of file DebugInterface.h.

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

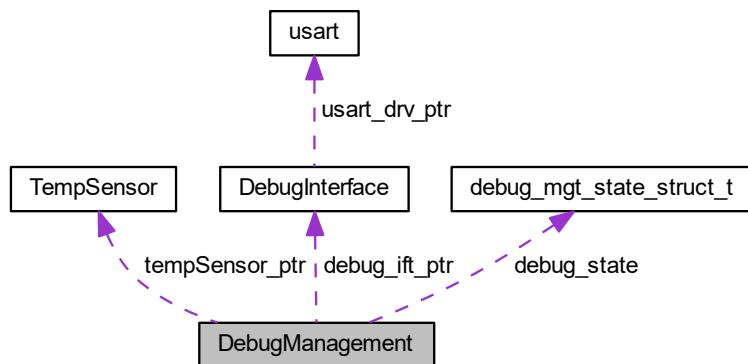
- [DebugInterface.h](#)
- [DebugInterface.cpp](#)

## 3.4 DebugManagement Class Reference

Debug management class.

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

Collaboration diagram for DebugManagement:



### Public Member Functions

- **DebugManagement ()**  
*Class constructor.*
- **void DisplayData ()**  
*Displays data on usart link.*
- **bool DebugModeManagement ()**  
*Management of debug mode.*
- **DebugInterface \* getIfItPtr ()**  
*Interface pointer get function.*
- **uint8\_t \* getMenuStringPtr ()**  
*Menu string get function.*
- **uint8\_t \* getInfoStringPtr ()**  
*Info string get function.*
- **void setInfoStringPtr (uint8\_t \*addr)**  
*Info message setting function.*

## Static Public Member Functions

- static void [DisplayPeriodicData\\_task \(\)](#)

*Displays periodic data on usart link.*

## Private Member Functions

- void [exitDebugMenu \(\)](#)  
*Debug menu exit function.*
- void [systemReset \(\)](#)  
*System reset function.*
- void [WatchdogMenuManagement \(uint8\\_t rcv\\_char\)](#)  
*Watchdog menu management function.*
- bool [MainMenuManagement \(uint8\\_t rcv\\_char\)](#)  
*Main menu management.*

## Private Attributes

- [DebugInterface \\* debug\\_ift\\_ptr](#)
- [TempSensor \\* tempSensor\\_ptr](#)
- [uint8\\_t \\* menu\\_string\\_ptr](#)
- [uint8\\_t \\* info\\_string\\_ptr](#)
- [debug\\_mgt\\_state\\_struct\\_t debug\\_state](#)

### 3.4.1 Detailed Description

Debug management class.

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

Definition at line 51 of file [DebugManagement.h](#).

### 3.4.2 Constructor & Destructor Documentation

### 3.4.2.1 DebugManagement()

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

Class constructor.

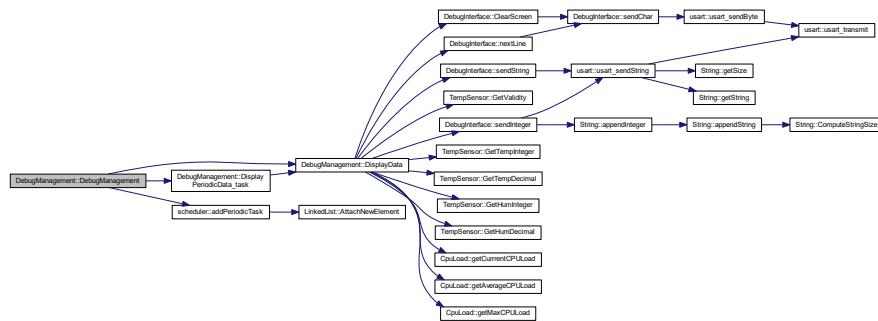
This function initializes the class. If needed, it creates a new instance of debug interface object.

#### Returns

Nothing

Definition at line 84 of file DebugManagement.cpp.

Here is the call graph for this function:



### 3.4.3 Member Function Documentation

#### 3.4.3.1 DebugModeManagement()

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

Management of debug mode.

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

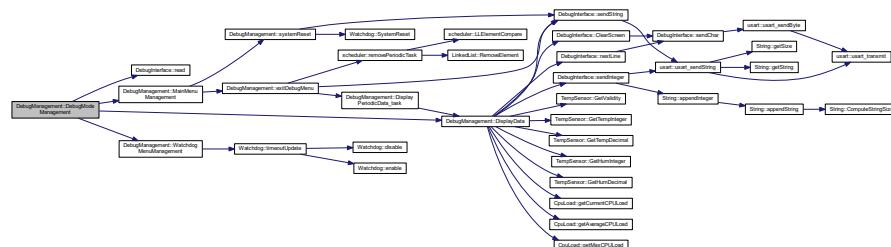
- MAIN\_MENU state : handles user choice in main menu and selects next state  
It is called each time a data is received on USART and debug mode is active.

**Returns**

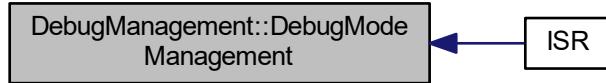
True if the debug mode shall be closed, false otherwise

Definition at line 194 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.4.3.2 DisplayData()**

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

Displays data on usart link.

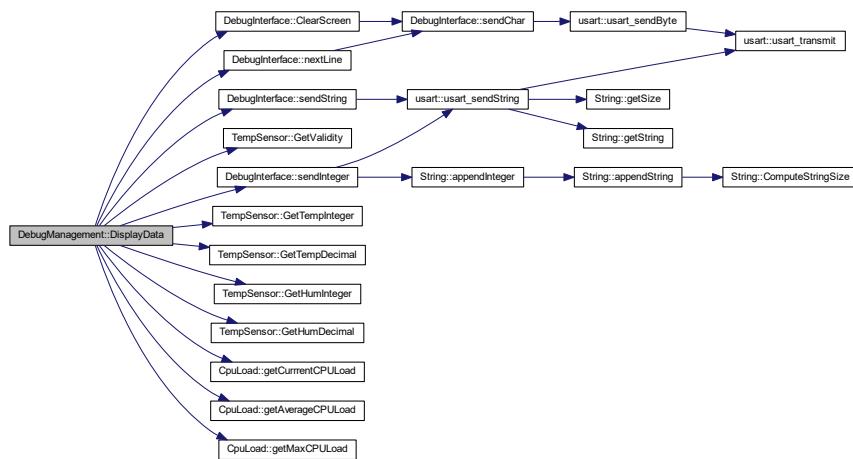
This task displays the menu and periodic data (temperature, humidity and CPU load) on usart screen.

**Returns**

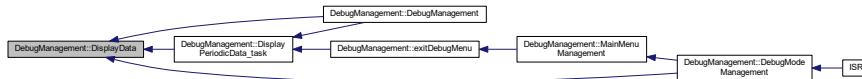
Nothing

Definition at line 112 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.4.3.3 DisplayPeriodicData\_task()

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

Displays periodic data on uart link.

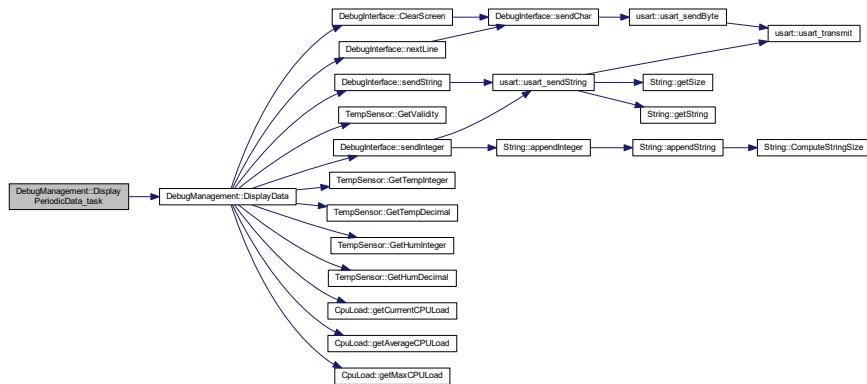
This task displays the menu and periodic data (temperature, humidity and CPU load) on uart screen. It only calls the function `DisplayData`.

#### Returns

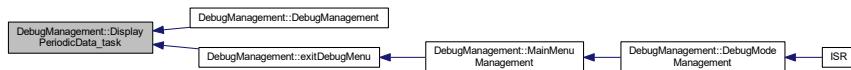
Nothing

Definition at line 186 of file `DebugManagement.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 3.4.3.4 exitDebugMenu()

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

Debug menu exit function.

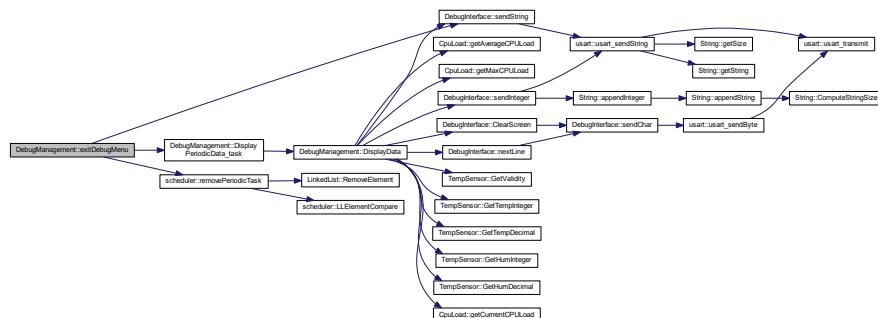
This function prepares the exit of debug menu. It writes the message "Bye !" on the screen and removes the periodic task from the scheduler.

##### Returns

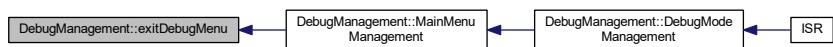
Nothing.

Definition at line 221 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.4.3.5 getIfpt()

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

Interface pointer get function.

This function returns the pointer to the debug interface object

#### Returns

Pointer to debug interface

Definition at line 94 of file DebugManagement.h.

### 3.4.3.6 getInfoStringPtr()

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

Info string get function.

This function returns the pointer to the info string to display

#### Returns

Info string pointer

Definition at line 114 of file DebugManagement.h.

### 3.4.3.7 getMenuStringPtr()

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

Menu string get function.

This function returns the pointer to the menu string to display

#### Returns

Menu string pointer

Definition at line 104 of file DebugManagement.h.

### 3.4.3.8 MainMenuManagement()

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

Main menu management.

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

#### Parameters

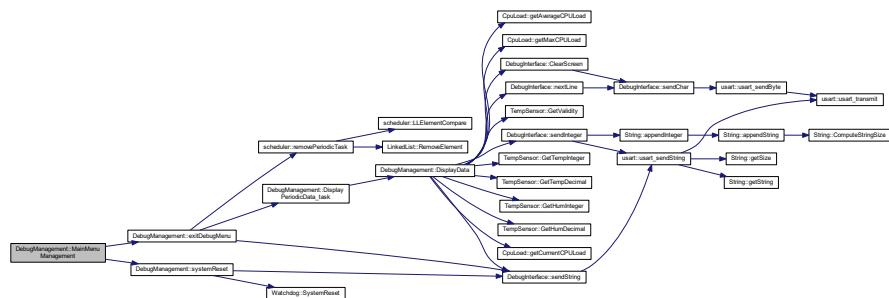
in	<i>rcv_char</i>	Character received on USART bus.
----	-----------------	----------------------------------

#### Returns

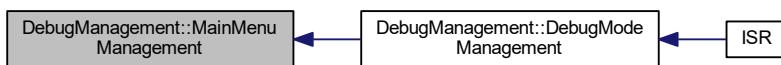
True if the debug mode shall be exited, false otherwise.

Definition at line 310 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.4.3.9 setInfoStringPtr()

```
void DebugManagement::setInfoStringPtr (
    uint8_t * addr ) [inline]
```

Info message setting function.

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

**Parameters**

in	<code>addr</code>	<code>String address</code>
----	-------------------	-----------------------------

**Returns**

Nothing

Definition at line 125 of file DebugManagement.h.

**3.4.3.10 systemReset()**

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

System reset function.

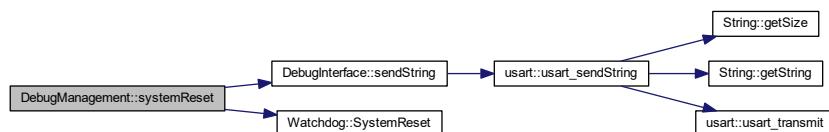
This function provokes a reset of the system. It displays a message on the screen and calls the reset function from watchdog class.

**Returns**

Nothing.

Definition at line 227 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.4.3.11 WatchdogMenuManagement()**

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

Watchdog menu management function.

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

**Parameters**

<code>in</code>	<code>rcv_char</code>	Character received on USART bus.
-----------------	-----------------------	----------------------------------

**Returns**

Nothing.

Definition at line 233 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 3.4.4 Member Data Documentation

### 3.4.4.1 debug\_ift\_ptr

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

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

Definition at line 132 of file DebugManagement.h.

### 3.4.4.2 debug\_state

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

Structure containing debug states for each menu

Definition at line 136 of file DebugManagement.h.

#### 3.4.4.3 info\_string\_ptr

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

Pointer to the info message to display

Definition at line 135 of file DebugManagement.h.

#### 3.4.4.4 menu\_string\_ptr

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

Pointer to the current menu string to display

Definition at line 134 of file DebugManagement.h.

#### 3.4.4.5 tempSensor\_ptr

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

Pointer to the temperature sensor object

Definition at line 133 of file DebugManagement.h.

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

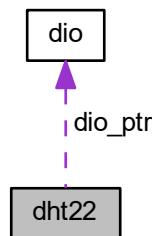
- [DebugManagement.h](#)
- [DebugManagement.cpp](#)

## 3.5 dht22 Class Reference

DHT 22 driver class.

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

Collaboration diagram for dht22:



## Public Member Functions

- [dht22](#) (uint8\_t port)  
*dht22 class constructor.*
- bool [read](#) (uint16\_t \*raw\_humidity, uint16\_t \*raw\_temperature)  
*Reads the data from DHT22.*

## Private Member Functions

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

## Private Attributes

- uint8\_t [dht22\\_port](#)
- [dio \\* dio\\_ptr](#)

### 3.5.1 Detailed Description

DHT 22 driver class.

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

Definition at line 19 of file dht22.h.

### 3.5.2 Constructor & Destructor Documentation

#### 3.5.2.1 [dht22\(\)](#)

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

[dht22](#) class constructor.

Initializes the class [dht22](#).

#### Parameters

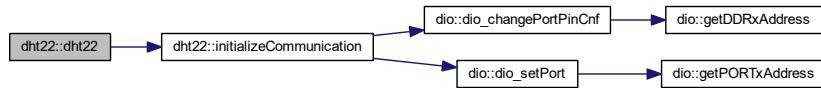
in	port	Encoded configuration of the port used for 1-wire communication.
----	------	--

#### Returns

Nothing

Definition at line 22 of file dht22.cpp.

Here is the call graph for this function:



### 3.5.3 Member Function Documentation

#### 3.5.3.1 initializeCommunication()

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

Initializes the communication.

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

##### Returns

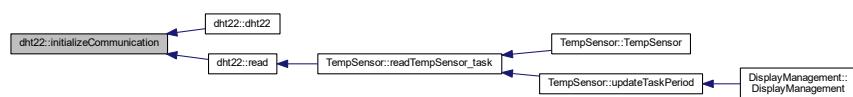
Nothing

Definition at line 201 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.5.3.2 `read()`

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

Reads the data from DHT22.

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

### Parameters

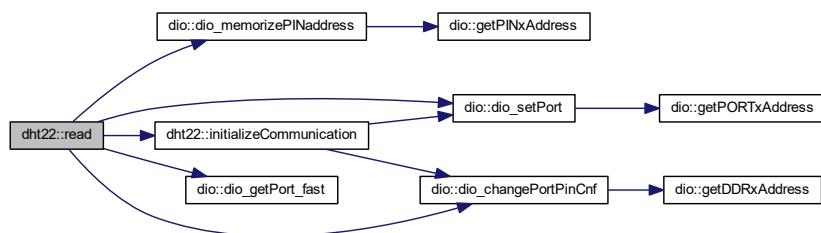
<code>out</code>	<code>raw_humidity</code>	Raw humidity value received from sensor
<code>out</code>	<code>raw_temperature</code>	Raw temperature value received from sensor

### Returns

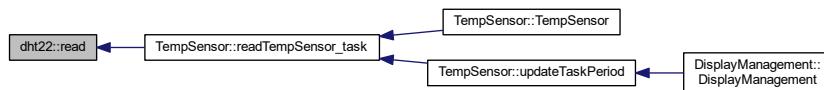
Validity of the read value

Definition at line 30 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 3.5.4 Member Data Documentation

### 3.5.4.1 dht22\_port

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

Variable containing the port used for 1-wire communication

Definition at line 45 of file dht22.h.

### 3.5.4.2 dio\_ptr

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

Pointer to the DIO object

Definition at line 46 of file dht22.h.

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

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

## 3.6 dio Class Reference

DIO class.

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

### Public Member Functions

- [\*\*dio \(\)\*\*](#)  
*dio class constructor*
- [\*\*void dio\\_setPort \(uint8\\_t portcode, bool state\)\*\*](#)  
*Port setting function.*
- [\*\*void dio\\_invertPort \(uint8\\_t portcode\)\*\*](#)  
*Inverts the state of output port.*
- [\*\*bool dio\\_getPort \(uint8\\_t portcode\)\*\*](#)  
*Gets the logical state of selected pin.*
- [\*\*bool dio\\_getPort\\_fast \(void\)\*\*](#)  
*Gets the logical state of the memorized pin.*
- [\*\*void dio\\_changePortPinCnf \(uint8\\_t portcode, uint8\\_t cnf\)\*\*](#)  
*Changes the IO configuration of the selected pin.*
- [\*\*void dio\\_memorizePINaddress \(uint8\\_t portcode\)\*\*](#)  
*Memorizes PINx register address and pin index.*

### Private Member Functions

- [\*\*void ports\\_init \(\)\*\*](#)  
*Digital ports hardware initialization function.*
- [\*\*uint8\\_t \\* getPORTxAddress \(uint8\\_t portcode\)\*\*](#)  
*Gets the physical address of the requested register PORTx.*
- [\*\*uint8\\_t \\* getPINxAddress \(uint8\\_t portcode\)\*\*](#)  
*Gets the physical address of the requested register PINx.*
- [\*\*uint8\\_t \\* getDDRxAddress \(uint8\\_t portcode\)\*\*](#)  
*Gets the physical address of the requested register DDRx.*

## Private Attributes

- `uint8_t * PINx_addr_mem`
- `uint8_t PINx_idx_mem`

### 3.6.1 Detailed Description

DIO class.

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

Definition at line 27 of file dio.h.

### 3.6.2 Constructor & Destructor Documentation

#### 3.6.2.1 dio()

`dio::dio ()`

`dio` class constructor

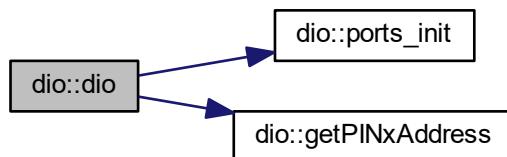
Initializes class `dio` and calls DIO hardware initialization function

Returns

Nothing

Definition at line 112 of file dio.cpp.

Here is the call graph for this function:



### 3.6.3 Member Function Documentation

#### 3.6.3.1 dio\_changePortPinCnf()

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

Changes the IO configuration of the selected pin.

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

**Parameters**

in	<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**

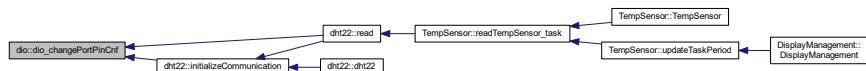
Nothing

Definition at line 149 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.6.3.2 dio\_getPort()**

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

Gets the logical state of selected pin.

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

**Parameters**

in	<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:



Here is the caller graph for this function:

**3.6.3.3 dio\_getPort\_fast()**

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

Gets the logical state of the memorized pin.

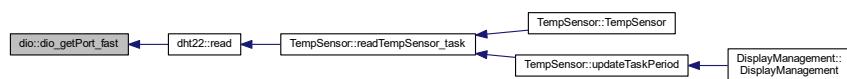
This function gets the logical value of the memorized pin. The corresponding port and pin index are stored in class members PINx\_addr\_mem and PINx\_idx\_mem. This mechanism is used to speed up reading time as this function no longer needs to extract register address and pin index from portcode.

**Returns**

Logical state of selected pin

Definition at line 171 of file dio.cpp.

Here is the caller graph for this function:



## 3.6.3.4 dio\_invertPort()

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

Inverts the state of output port.

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

## Parameters

in	<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:



## 3.6.3.5 dio\_memorizePINaddress()

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

Memorizes PINx register address and pin index.

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

**Parameters**

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

**Returns**

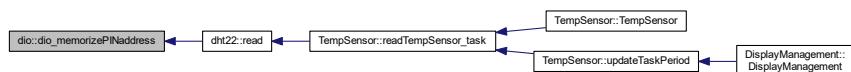
Nothing

Definition at line 165 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.6.3.6 dio\_setPort()**

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

Port setting function.

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

**Parameters**

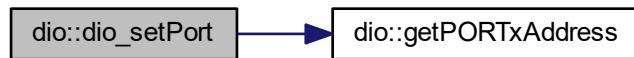
in	<i>portcode</i>	Encoded pin and register index
in	<i>state</i>	Requested state to set pin

**Returns**

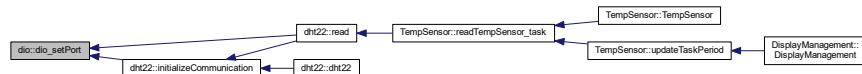
Nothing

Definition at line 121 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.6.3.7 getDDRxAddress()**

```

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

```

Gets the physical address of the requested register DDRx.

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

**Parameters**

in	<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:



### 3.6.3.8 getPINxAddress()

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

Gets the physical address of the requested register PINx.

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

#### Parameters

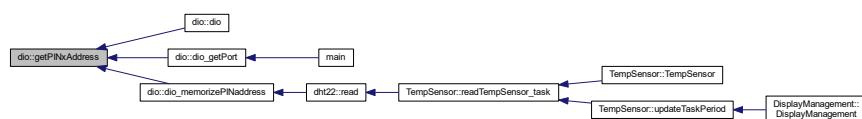
in	<i>portcode</i>	Encoded port code
----	-----------------	-------------------

#### Returns

Pointer to the PINx register

Definition at line 54 of file dio.cpp.

Here is the caller graph for this function:



### 3.6.3.9 getPORTxAddress()

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

Gets the physical address of the requested register PORTx.

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

#### Parameters

in	<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:



### 3.6.3.10 ports\_init()

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

Digital ports hardware initialization function.

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

**Returns**

Nothing

Definition at line 16 of file dio.cpp.

Here is the caller graph for this function:



## 3.6.4 Member Data Documentation

### 3.6.4.1 PINx\_addr\_mem

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

Memorizes physical address of register PINx in order to speed up register reading time in function dio\_getPort\_fast

Definition at line 142 of file dio.h.

### 3.6.4.2 PINx\_idx\_mem

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

Memorizes pin index of register PINx in order to speed up register reading time in function dio\_getPort\_fast

Definition at line 143 of file dio.h.

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

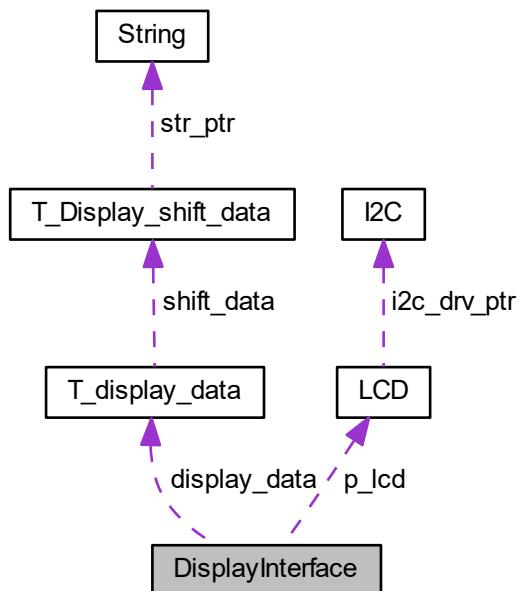
- [dio.h](#)
- [dio.cpp](#)

## 3.7 DisplayInterface Class Reference

Display interface services class.

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

Collaboration diagram for DisplayInterface:



## Public Member Functions

- `DisplayInterface (const T_LCD_conf_struct *LCD_init_cnf)`  
*Class constructor.*
- `bool DisplayFullLine (uint8_t *str, uint8_t size, uint8_t line, T_DisplayInterface_LineDisplayMode mode=NORMAL, T_DisplayInterface_LineAlignment alignment=LEFT)`  
*Line display function.*
- `bool ClearLine (uint8_t line)`  
*Line clearing function.*
- `void ClearFullScreen ()`  
*Screen cleaning function.*
- `bool IsLineEmpty (uint8_t line)`  
*Empty line get function.*
- `T_display_data * getDisplayDataPtr ()`  
*Display data get function.*
- `void setLineAlignmentAndRefresh (uint8_t line, T_DisplayInterface_LineAlignment alignment)`  
*Text alignment function.*

## Static Public Member Functions

- `static void shiftLine_task ()`  
*Line shifting periodic task.*

## Private Member Functions

- `uint8_t FindFirstCharAddr (uint8_t line)`  
*Finds start address of a line.*
- `void RefreshLine (uint8_t line)`  
*Line refresh function.*
- `void ClearStringInDataStruct (uint8_t line)`  
*String data clearing structure.*
- `void setLineAlignment (uint8_t line)`  
*Text alignment setting function.*
- `void updateLineAndRefresh (uint8_t *str, uint8_t size, uint8_t line)`  
*Line data string update function.*

## Private Attributes

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

### 3.7.1 Detailed Description

Display interface services class.

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

Definition at line 76 of file DisplayInterface.h.

### 3.7.2 Constructor & Destructor Documentation

#### 3.7.2.1 DisplayInterface()

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

Class constructor.

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

##### Parameters

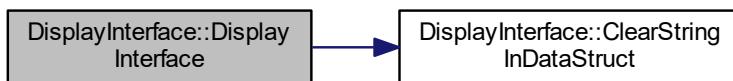
in	<i>LCD_init_cnf</i>	Initial configuration of the screen
----	---------------------	-------------------------------------

##### Returns

Nothing

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

Here is the call graph for this function:



### 3.7.3 Member Function Documentation

#### 3.7.3.1 ClearFullScreen()

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

Screen cleaning function.

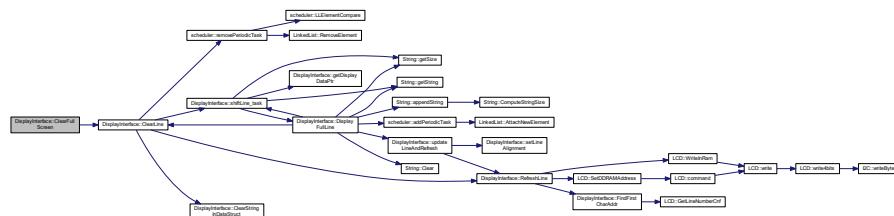
This functions clears the entire display. It uses the `ClearLine` function on every line of the screen.

**Returns**

Nothing

Definition at line 265 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.7.3.2 ClearLine()**

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

Line clearing function.

This function clears the requested line. It sets the corresponding DDRAM addresses to the ASCII value of space character. If it was the last line with a display shift in progress, it removes the periodic task from the scheduler.

**Parameters**

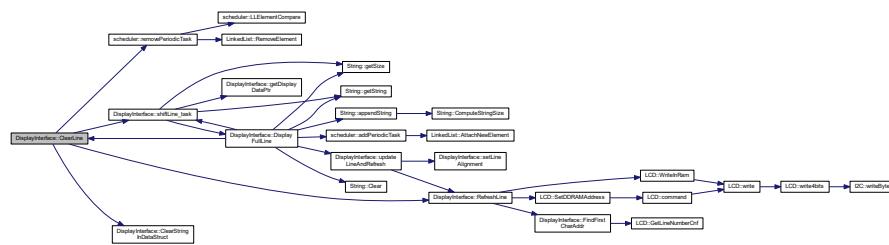
in	<i>line</i>	Line to clear
----	-------------	---------------

**Returns**

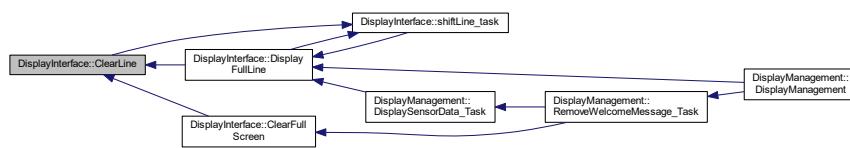
True if the line has been cleared, false otherwise

Definition at line 221 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.7.3.3 ClearStringInDataStruct()

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

**String** data clearing structure.

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

#### Parameters

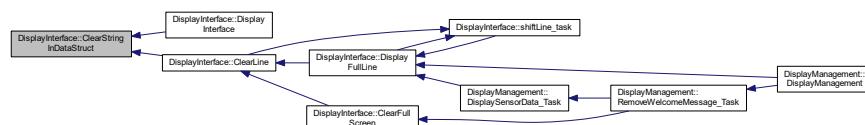
in	<i>line</i>	Line to clear
----	-------------	---------------

#### Returns

Nothing

Definition at line 174 of file DisplayInterface.cpp.

Here is the caller graph for this function:



### 3.7.3.4 DisplayFullLine()

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

Line display function.

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

#### Parameters

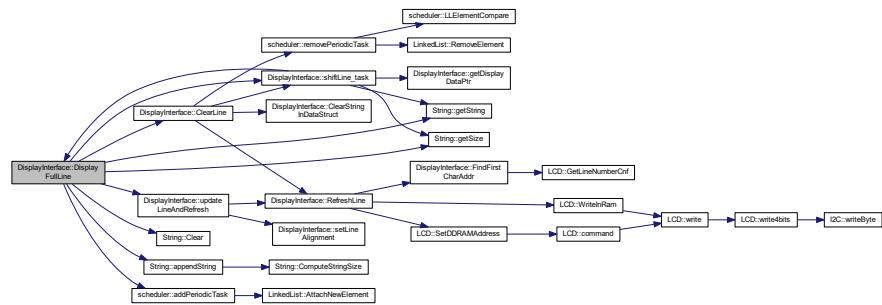
in	<b>str</b>	Pointer to the string to display
in	<b>size</b>	Size of the string to display
in	<b>line</b>	Index of the line where the string shall be displayed
in	<b>mode</b>	Display mode
in	<b>alignment</b>	Requested alignment for the line

#### Returns

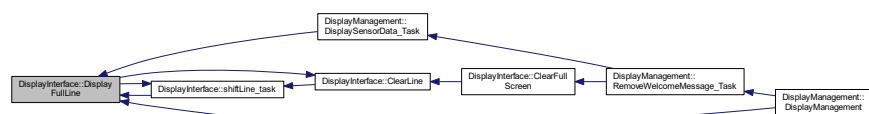
True if the line has been correctly displayed, false otherwise

Definition at line 59 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.7.3.5 FindFirstCharAddr()

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

Finds start address of a line.

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

#### Parameters

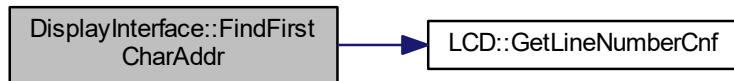
<code>in</code>	<code>line</code>	Line which address shall be found
-----------------	-------------------	-----------------------------------

#### Returns

Address in DDRAM of the first character of the line

Definition at line 182 of file `DisplayInterface.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.7.3.6 getDisplayDataPtr()

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

Display data get function.

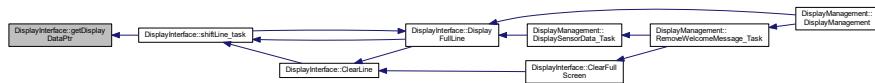
This function returns a pointer to the display data structure.

**Returns**

Pointer to display data structure.

Definition at line 142 of file DisplayInterface.h.

Here is the caller graph for this function:

**3.7.3.7 IsLineEmpty()**

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

Empty line get function.

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

**Parameters**

in	<i>line</i>	Requested line
----	-------------	----------------

**Returns**

True if the line is empty, false otherwise

Definition at line 273 of file DisplayInterface.cpp.

**3.7.3.8 RefreshLine()**

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

Line refresh function.

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

**Parameters**

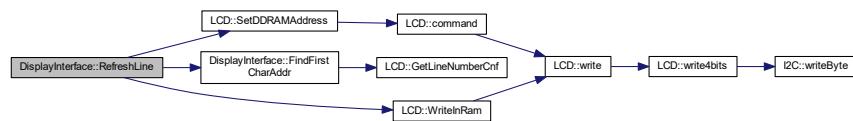
in	<i>line</i>	Line to refresh
----	-------------	-----------------

**Returns**

Nothing

Definition at line 161 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.7.3.9 setLineAlignment()**

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

Text alignment setting function.

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

**Parameters**

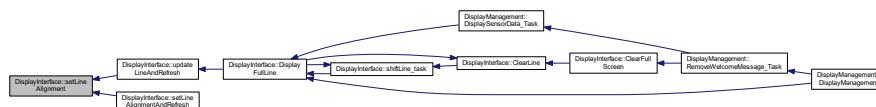
in	<i>line</i>	Line to update
----	-------------	----------------

**Returns**

Nothing

Definition at line 330 of file DisplayInterface.cpp.

Here is the caller graph for this function:



### 3.7.3.10 setLineAlignmentAndRefresh()

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

Text alignment function.

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

#### Parameters

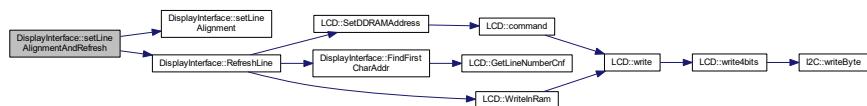
in	<i>line</i>	Requested line to update
in	<i>alignment</i>	Requested alignment for the text

#### Returns

Nothing

Definition at line 448 of file `DisplayInterface.cpp`.

Here is the call graph for this function:



### 3.7.3.11 shiftLine\_task()

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

Line shifting periodic task.

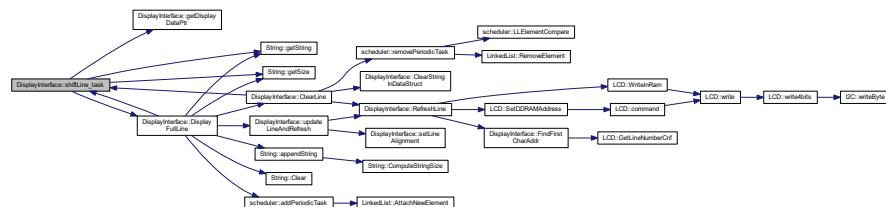
This function is called periodically by the scheduler. It shifts all the lines in line shifting mode and updates the data structures.

**Returns**

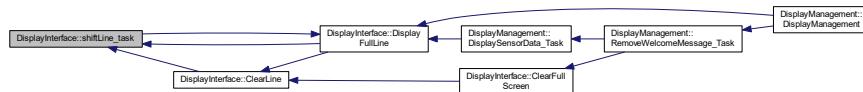
Nothing

Definition at line 282 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.7.3.12 updateLineAndRefresh()**

```

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

Line data string update function.

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

**Parameters**

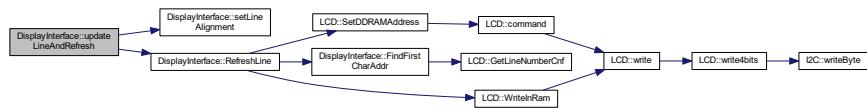
<b>in</b>	<b>str</b>	Pointer to the string to display
<b>in</b>	<b>size</b>	Size of the string
<b>in</b>	<b>line</b>	Line to update

**Returns**

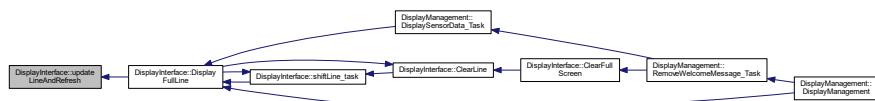
Nothing

Definition at line 145 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 3.7.4 Member Data Documentation

### 3.7.4.1 display\_data

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

Screen display data

Definition at line 165 of file `DisplayInterface.h`.

### 3.7.4.2 dummy

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

Needed for data alignment

Definition at line 164 of file `DisplayInterface.h`.

### 3.7.4.3 isShiftInProgress

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

Flag indicating if a shift is in progress on any line

Definition at line 166 of file `DisplayInterface.h`.

### 3.7.4.4 p\_lcd

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

Pointer to the attached LCD driver object

Definition at line 163 of file DisplayInterface.h.

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

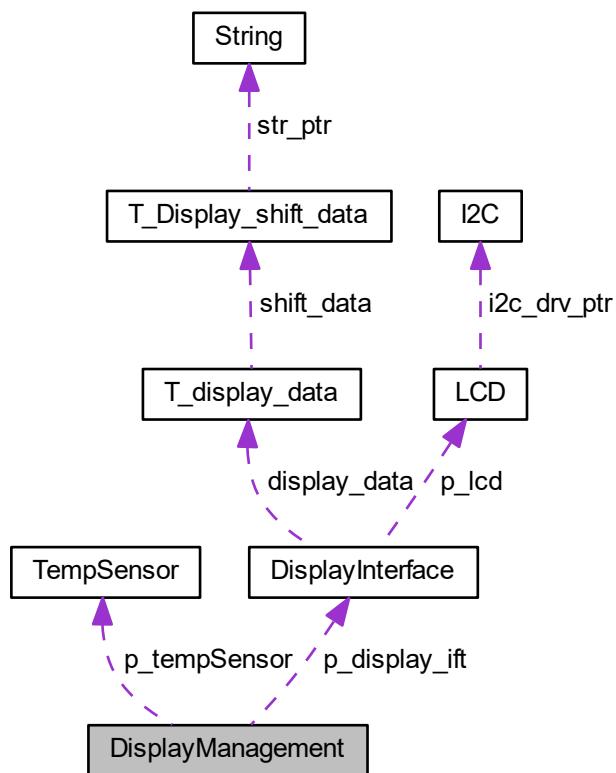
- [DisplayInterface.h](#)
- [DisplayInterface.cpp](#)

## 3.8 DisplayManagement Class Reference

Display management class.

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

Collaboration diagram for DisplayManagement:



## Public Member Functions

- [DisplayManagement \(\)](#)  
*Class constructor.*
- [DisplayInterface \\* GetIfpPointer \(\)](#)  
*Interface pointer get function.*
- [TempSensor \\* GetTempSensorPtr \(\)](#)  
*Sensor pointer get function.*

## Static Public Member Functions

- [static void DisplaySensorData\\_Task \(\)](#)  
*Periodic task for displaying sensor data.*
- [static void RemoveWelcomeMessage\\_Task \(\)](#)  
*End of welcome message task.*

## Private Attributes

- [DisplayInterface \\* p\\_display\\_ift](#)
- [TempSensor \\* p\\_tempSensor](#)

### 3.8.1 Detailed Description

Display management class.

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

Definition at line 53 of file `DisplayManagement.h`.

### 3.8.2 Constructor & Destructor Documentation

#### 3.8.2.1 [DisplayManagement\(\)](#)

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

Class constructor.

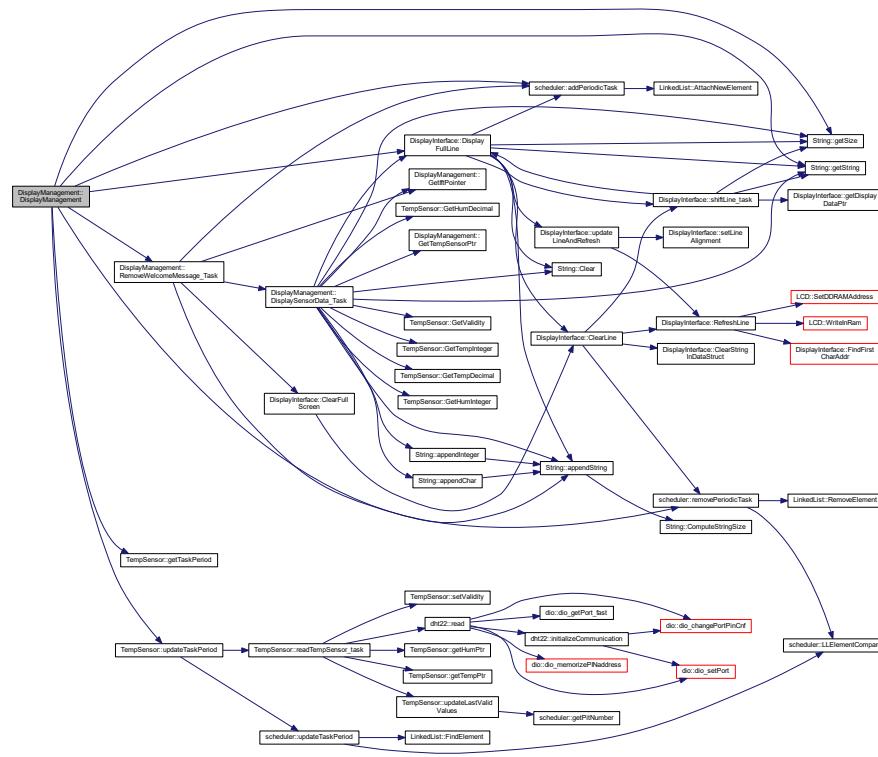
This class initializes display management.  
It creates a display interface object and initializes all class variables.

**Returns**

Nothing

Definition at line 30 of file DisplayManagement.cpp.

Here is the call graph for this function:



### 3.8.3 Member Function Documentation

#### 3.8.3.1 DisplaySensorData\_Task()

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

Periodic task for displaying sensor data.

This function displays the sensors data on the screen. Currently temperature and humidity data coming from [dht22](#) sensor are displayed.

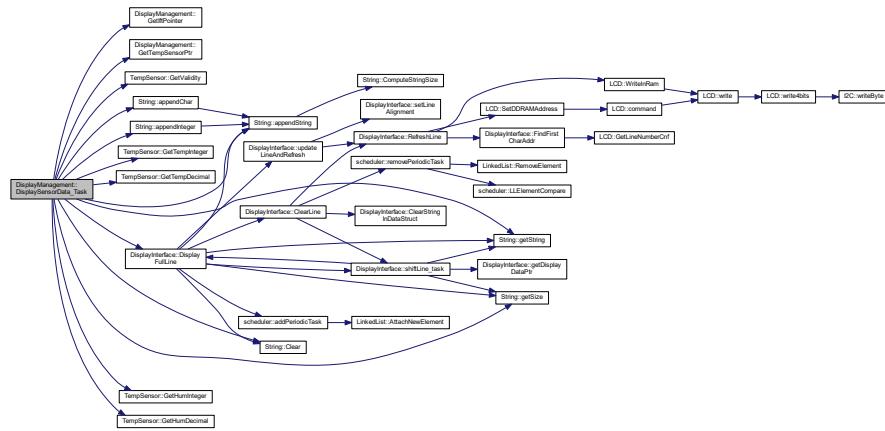
It is called periodically by scheduler.

**Returns**

Nothing

Definition at line 74 of file DisplayManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.8.3.2 GetIftPointer()**

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

Interface pointer get function.

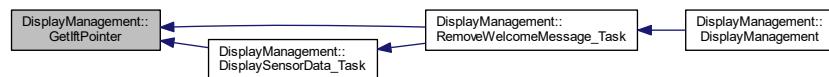
This function returns the pointer to the display interface object

**Returns**

Pointer to display interface object

Definition at line 82 of file DisplayManagement.h.

Here is the caller graph for this function:



### 3.8.3.3 GetTempSensorPtr()

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

Sensor pointer get function.

This function returns the pointer to the temperature sensor object

#### Returns

Pointer to sensor object

Definition at line 93 of file DisplayManagement.h.

Here is the caller graph for this function:



### 3.8.3.4 RemoveWelcomeMessage\_Task()

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

End of welcome message task.

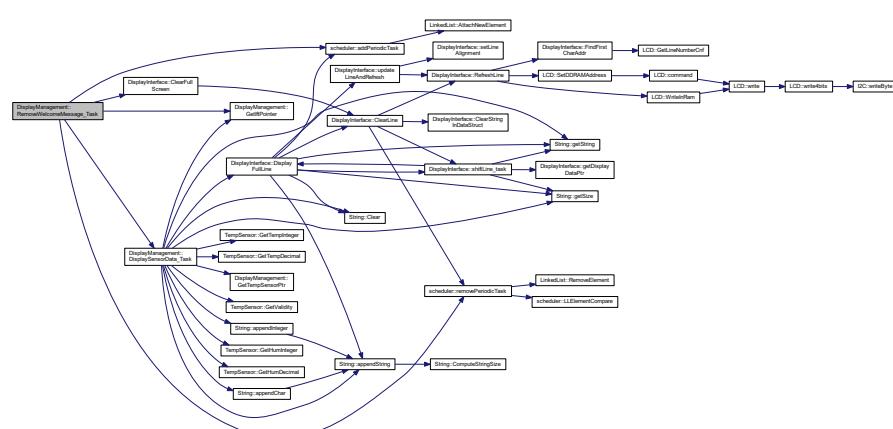
This task clears the welcome message from the screen and start periodic display of sensor data. This task shall be added in scheduler when the welcome message is displayed on screen. As it shall be called only once, the task removes itself from the scheduler after the first call.

#### Returns

Nothing

Definition at line 57 of file DisplayManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.8.4 Member Data Documentation

#### 3.8.4.1 p\_display\_ift

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

Pointer to the display interface object

Definition at line 110 of file DisplayManagement.h.

#### 3.8.4.2 p\_tempSensor

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

Pointer to the temperature sensor object

Definition at line 111 of file DisplayManagement.h.

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

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

## 3.9 I2C Class Reference

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

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

## Public Member Functions

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

## Private Member Functions

- `void initializeBus ()`  
*I2C bus initialization.*

## Private Attributes

- `uint8_t tx_address`
- `uint32_t bitrate`

### 3.9.1 Detailed Description

Two-wire serial interface (`I2C`) class definition.

This class manages `I2C` driver.

Definition at line 23 of file I2C.h.

### 3.9.2 Constructor & Destructor Documentation

#### 3.9.2.1 `I2C()`

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

`I2C` class constructor.

This function initializes the `I2C` class and calls bus initialization function

#### Parameters

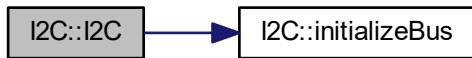
in	<code>l_bitrate</code>	Requested bitrate for <code>I2C</code> bus (in Hz)
----	------------------------	--

Returns

Nothing

Definition at line 16 of file I2C.cpp.

Here is the call graph for this function:



### 3.9.3 Member Function Documentation

#### 3.9.3.1 initializeBus()

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

I2C bus initialization.

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

Returns

Nothing

Definition at line 77 of file I2C.cpp.

Here is the caller graph for this function:



#### 3.9.3.2 setBitRate()

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

Variable bitrate setting function.

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

**Parameters**

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

**Returns**

Nothing

Definition at line 72 of file I2C.cpp.

**3.9.3.3 setTxAddress()**

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

Setting function for Tx **I2C** address.

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

**Parameters**

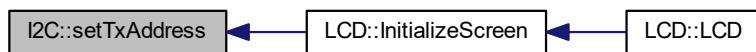
in	<i>address</i>	Requested Tx address
----	----------------	----------------------

**Returns**

Nothing

Definition at line 67 of file I2C.cpp.

Here is the caller graph for this function:

**3.9.3.4 writeByte()**

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

Byte sending function.

This function sends one byte on **I2C** bus

**Parameters**

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

**Returns**

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

Definition at line 24 of file I2C.cpp.

Here is the caller graph for this function:



### 3.9.4 Member Data Documentation

#### 3.9.4.1 bitrate

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

Definition at line 63 of file I2C.h.

#### 3.9.4.2 tx\_address

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

Definition at line 62 of file I2C.h.

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

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

## 3.10 keepAliveLed Class Reference

Class for keep-alive LED blinking.

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

## Public Member Functions

- [keepAliveLed \(\)](#)

*Class constructor.*

## Static Public Member Functions

- [static void blinkLed\\_task \(\)](#)

*Task for LED blinking.*

### 3.10.1 Detailed Description

Class for keep-alive LED blinking.

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

Definition at line 22 of file `keepAliveLed.h`.

### 3.10.2 Constructor & Destructor Documentation

#### 3.10.2.1 `keepAliveLed()`

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

Class constructor.

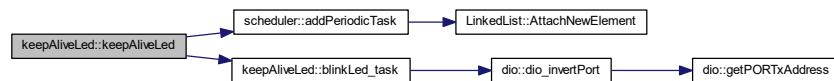
This function initializes the class `keepAliveLed`

#### Returns

Nothing

Definition at line 22 of file `keepAliveLed.cpp`.

Here is the call graph for this function:



### 3.10.3 Member Function Documentation

### 3.10.3.1 blinkLed\_task()

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

Task for LED blinking.

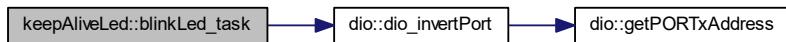
This function is inserted into the scheduler. It changes the state of the LED output to make it blink

#### Returns

Nothing

Definition at line 28 of file `keepAliveLed.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



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

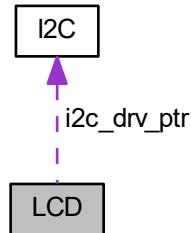
- [keepAliveLed.h](#)
- [keepAliveLed.cpp](#)

## 3.11 LCD Class Reference

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

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

Collaboration diagram for LCD:



## Public Member Functions

- `LCD (const T_LCD_conf_struct *init_conf)`  
*LCD class constructor.*
- `void command (T_LCD_command cmd)`  
*LCD command management function.*
- `void ConfigureBacklight (bool enable)`  
*Backlight configuration function.*
- `void ConfigureLineNumber (bool param)`  
*Line type configuration function.*
- `void ConfigureFontType (bool param)`  
*Font configuration function.*
- `void ConfigureDisplayOnOff (bool param)`  
*Display configuration function.*
- `void ConfigureCursorOnOff (bool param)`  
*Cursor configuration function.*
- `void ConfigureCursorBlink (bool param)`  
*Cursor blinking configuration function.*
- `void ConfigureEntryModeDir (bool param)`  
*Entry mode direction configuration function.*
- `void ConfigureEntryModeShift (bool param)`  
*Entry mode shift configuration function.*
- `void ConfigureI2CAddr (uint8_t param)`  
*I2C address configuration function.*
- `void SetDDRAMAddress (uint8_t addr)`  
*DDRAM address setting function.*
- `uint8_t GetDDRAMAddress ()`  
*DDRAM address get function.*
- `void WriteInRam (uint8_t a_char, T_LCD_ram_area area)`  
*Screen RAM write function.*
- `bool GetLineNumberCnf ()`  
*Number of line get function.*

## Private Member Functions

- void `write4bits` (uint8\_t data)  
*I2C write function for 4-bits mode.*
- void `write` (uint8\_t data, `T_LCD_config_mode` mode)  
*I2C write function.*
- void `InitializeScreen` ()  
*Screen configuration function.*

## Private Attributes

- bool `backlight_enable`
- bool `cnfLineNumber`
- bool `cnfFontType`
- bool `cnfDisplayOnOff`
- bool `cnfCursorOnOff`
- bool `cnfCursorBlink`
- bool `cnfEntryModeDir`
- bool `cnfEntryModeShift`
- uint8\_t `cnfI2C_addr`
- I2C \* `i2c_drv_ptr`
- uint8\_t `ddram_addr`

### 3.11.1 Detailed Description

Class for `LCD` S2004A display driver.

This class handles functions managing `LCD` display S2004a on `I2C` bus

Definition at line 147 of file `LCD.h`.

### 3.11.2 Constructor & Destructor Documentation

#### 3.11.2.1 LCD()

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

`LCD` class constructor.

This constructor function initializes the class `LCD` and calls screen configuration function. It also creates a new instance of the `I2C` driver if needed.

#### Parameters

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

**Returns**

Nothing

Definition at line 18 of file LCD.cpp.

Here is the call graph for this function:



### 3.11.3 Member Function Documentation

#### 3.11.3.1 command()

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

**LCD** command management function.

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

**Parameters**

in	cmd	Requested command
----	-----	-------------------

**Returns**

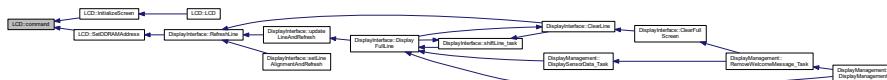
Nothing

Definition at line 125 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.11.3.2 ConfigureBacklight()

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

Backlight configuration function.

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

#### Parameters

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

#### Returns

Nothing

Definition at line 178 of file LCD.h.

Here is the caller graph for this function:



### 3.11.3.3 ConfigureCursorBlink()

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

Cursor blinking configuration function.

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

#### Parameters

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

#### Returns

Nothing

Definition at line 238 of file LCD.h.

Here is the caller graph for this function:



### 3.11.3.4 ConfigureCursorOnOff()

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

Cursor configuration function.

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

#### Parameters

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

#### Returns

Nothing

Definition at line 226 of file LCD.h.

Here is the caller graph for this function:



### 3.11.3.5 ConfigureDisplayOnOff()

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

Display configuration function.

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

#### Parameters

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

#### Returns

Nothing

Definition at line 214 of file LCD.h.

Here is the caller graph for this function:



### 3.11.3.6 ConfigureEntryModeDir()

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

Entry mode direction configuration function.

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

**Parameters**

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

**Returns**

Nothing

Definition at line 250 of file LCD.h.

Here is the caller graph for this function:



### 3.11.3.7 ConfigureEntryModeShift()

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

Entry mode shift configuration function.

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

**Parameters**

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

**Returns**

Nothing

Definition at line 262 of file LCD.h.

Here is the caller graph for this function:



### 3.11.3.8 ConfigureFontType()

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

Font configuration function.

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

**Parameters**

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

**Returns**

Nothing

Definition at line 202 of file LCD.h.

Here is the caller graph for this function:



### 3.11.3.9 ConfigureI2CAddr()

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

I2C address configuration function.

This function configures the I2V address of the [LCD](#) screen according to the parameter.

#### Parameters

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

#### Returns

Nothing

Definition at line 274 of file LCD.h.

Here is the caller graph for this function:



### 3.11.3.10 ConfigureLineNumber()

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

Line type configuration function.

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

#### Parameters

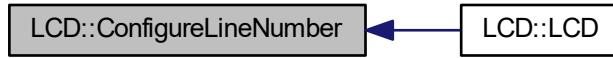
in	<i>param</i>	Configuration value
----	--------------	---------------------

#### Returns

Nothing

Definition at line 190 of file LCD.h.

Here is the caller graph for this function:



### 3.11.3.11 GetDDRAMAddress()

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

DDRAM address get function.

This function return the value of the current DDRAM address stored in internal variable ddrum\_addr.

#### Returns

Current DDRAM address

Definition at line 294 of file LCD.h.

### 3.11.3.12 GetLineNumberCnf()

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

Number of line get function.

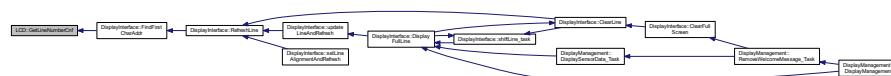
This function returns the line number configuration of the screen : 1 or 2 lines mode.

#### Returns

Line number configuration

Definition at line 316 of file LCD.h.

Here is the caller graph for this function:



### 3.11.3.13 InitializeScreen()

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

Screen configuration function.

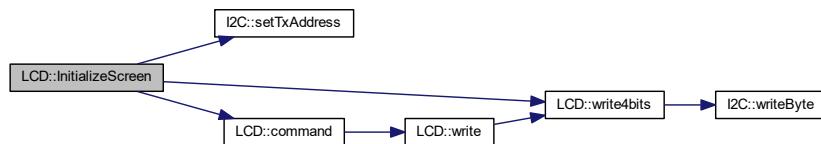
This function configures the [LCD](#) screen. It's must be called during initialization phase, or the screen won't be usable. The configuration process is described in [LCD](#) datasheet J2004A-GFDN-DYNC

#### Returns

Nothing

Definition at line 73 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.11.3.14 SetDDRAMAddress()

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

DDRAM address setting function.

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

**Parameters**

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

**Returns**

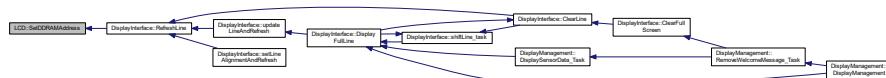
Nothing

Definition at line 168 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.11.3.15 write()**

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

**I2C** write function.

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

**Parameters**

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

**Returns**

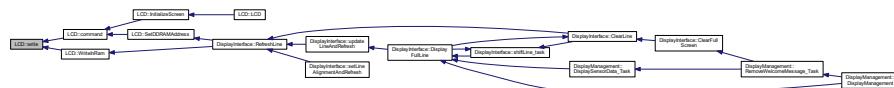
Nothing

Definition at line 62 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.11.3.16 write4bits()

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

I2C write function for 4-bits mode.

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

#### Parameters

in	<i>data</i>	8-bit data to send
----	-------------	--------------------

#### Returns

Nothing

Definition at line 45 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.11.3.17 WriteInRam()

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

Screen RAM write function.

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

#### Parameters

in	<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 190 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.11.4 Member Data Documentation

### 3.11.4.1 **backlight\_enable**

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

Backlight enable flag

Definition at line 324 of file LCD.h.

### 3.11.4.2 **cnfCursorBlink**

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

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

Definition at line 329 of file LCD.h.

### 3.11.4.3 **cnfCursorOnOff**

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

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

Definition at line 328 of file LCD.h.

### 3.11.4.4 **cnfDisplayOnOff**

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

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

Definition at line 327 of file LCD.h.

### 3.11.4.5 **cnfEntryModeDir**

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

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

Definition at line 330 of file LCD.h.

#### 3.11.4.6 cnfEntryModeShift

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

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

Definition at line 331 of file LCD.h.

#### 3.11.4.7 cnfFontType

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

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

Definition at line 326 of file LCD.h.

#### 3.11.4.8 cnfI2C\_addr

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

I2C address of the [LCD](#) screen

Definition at line 332 of file LCD.h.

#### 3.11.4.9 cnfLineNumber

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

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

Definition at line 325 of file LCD.h.

#### 3.11.4.10 ddram\_addr

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

Screen DDRAM address

Definition at line 336 of file LCD.h.

### 3.11.4.11 i2c\_drv\_ptr

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

Pointer to the [I2C](#) driver object

Definition at line 334 of file LCD.h.

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

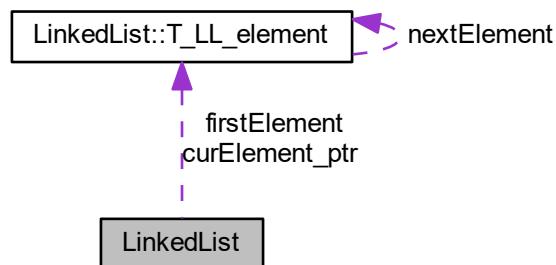
- [LCD.h](#)
- [LCD.cpp](#)

## 3.12 LinkedList Class Reference

Linked list class.

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

Collaboration diagram for [LinkedList](#):



## Classes

- struct [T\\_LL\\_element](#)

*Type defining a linked list element.*

## Public Member Functions

- [LinkedList \(\)](#)  
*Class constructor.*
- [~LinkedList \(\)](#)  
*Class destructor.*
- [void AttachNewElement \(void \\*data\\_ptr\)](#)  
*Add an new element to the list.*
- [bool RemoveElement \(CompareFctPtr\\_t comparisonFct\\_ptr, void \\*reference\\_ptr\)](#)  
*Removes an element from the chain.*
- [void \\*getCurrentElement \(\)](#)  
*Current element get function.*
- [bool MoveToNextElement \(\)](#)  
*Move to next element function.*
- [void ResetElementPtr \(\)](#)  
*Resets element pointer.*
- [bool IsLLEmpty \(\)](#)  
*Empty linked list.*
- [bool FindElement \(CompareFctPtr\\_t comparisonFct\\_ptr, void \\*reference\\_ptr, void \\*\\*chainElement\\_ptr\)](#)  
*Element finding function.*

## Private Types

- [typedef struct LinkedList::T\\_LL\\_element T\\_LL\\_element](#)  
*Type defining a linked list element.*

## Private Attributes

- [T\\_LL\\_element \\* firstElement](#)
- [T\\_LL\\_element \\* curElement\\_ptr](#)

### 3.12.1 Detailed Description

Linked list class.

This class defines a linked list and the associated services.

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

Definition at line 22 of file [LinkedList.h](#).

### 3.12.2 Member Typedef Documentation

### 3.12.2.1 T\_LL\_element

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

Type defining a linked list element.

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

## 3.12.3 Constructor & Destructor Documentation

### 3.12.3.1 LinkedList()

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

Class constructor.

This constructor initializes a linked list

Returns

Nothing

Definition at line 37 of file LinkedList.cpp.

### 3.12.3.2 ~LinkedList()

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

Class destructor.

This function deletes the linked list

Returns

Nothing

Definition at line 43 of file LinkedList.cpp.

Here is the call graph for this function:



### 3.12.4 Member Function Documentation

#### 3.12.4.1 AttachNewElement()

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

Add an new element to the list.

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

##### Parameters

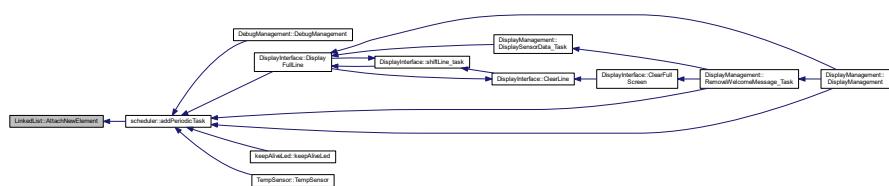
in	<i>data_ptr</i>	Pointer to the data element
----	-----------------	-----------------------------

##### Returns

Nothing

Definition at line 61 of file LinkedList.cpp.

Here is the caller graph for this function:



#### 3.12.4.2 FindElement()

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

Element finding function.

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

**Parameters**

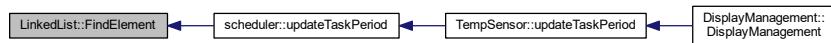
in	<i>comparisonFct_ptr</i>	Pointer to the comparison function
in	<i>reference_ptr</i>	Pointer to the element to find in the chain
out	<i>chainElement_ptr</i>	Pointer to pointer to the found element

**Returns**

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

Definition at line 152 of file `LinkedList.cpp`.

Here is the caller graph for this function:

**3.12.4.3 getCurrentElement()**

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

Current element get function.

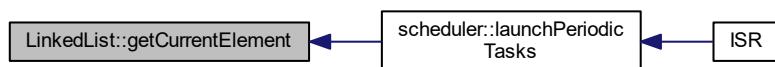
This function returns a pointer to the current pointed data in the chain.

**Returns**

Pointer to the current data

Definition at line 67 of file `LinkedList.h`.

Here is the caller graph for this function:



#### 3.12.4.4 IsLLEmpty()

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

Empty linked list.

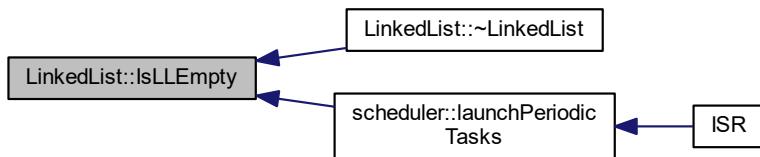
This function checks whether the linked list is empty or not (pointer to first element is equal to 0 or not).

##### Returns

True if the list is empty, false otherwise

Definition at line 144 of file LinkedList.cpp.

Here is the caller graph for this function:



#### 3.12.4.5 MoveToNextElement()

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

Move to next element function.

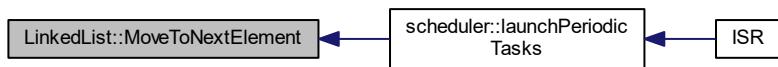
This function moves the element pointer to the next element of the chain.

##### Returns

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

Definition at line 130 of file LinkedList.cpp.

Here is the caller graph for this function:



### 3.12.4.6 RemoveElement()

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

Removes an element from the chain.

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

#### Parameters

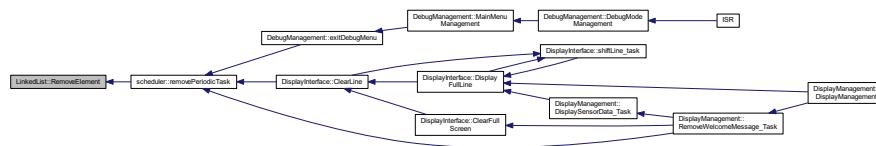
in	<i>comparisonFct_ptr</i>	Pointer to the comparison function to use
in	<i>reference_ptr</i>	Pointer to the reference data used for comparison

#### Returns

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

Definition at line 86 of file LinkedList.cpp.

Here is the caller graph for this function:



### 3.12.4.7 ResetElementPtr()

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

Resets element pointer.

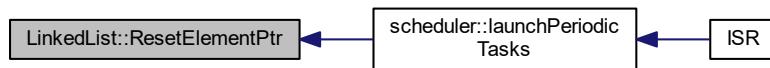
This function sets the element pointer to the first element of the chain.

#### Returns

Nothing

Definition at line 86 of file LinkedList.h.

Here is the caller graph for this function:



### 3.12.5 Member Data Documentation

#### 3.12.5.1 curElement\_ptr

`T_LL_element* LinkedList::curElement_ptr [private]`

Pointer to the current element of the list

Definition at line 125 of file `LinkedList.h`.

#### 3.12.5.2 firstElement

`T_LL_element* LinkedList::firstElement [private]`

Pointer to the first element of the list

Definition at line 124 of file `LinkedList.h`.

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

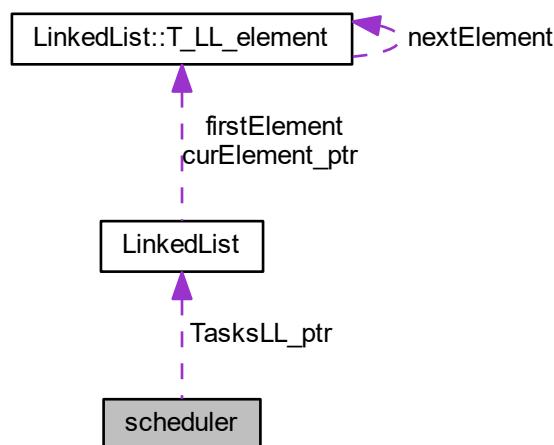
- [LinkedList.h](#)
- [LinkedList.cpp](#)

## 3.13 scheduler Class Reference

Scheduler class.

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

Collaboration diagram for scheduler:



## Classes

- struct [Task\\_t](#)  
*Type defining a task structure.*

## Public Member Functions

- [scheduler \(\)](#)  
*scheduler class constructor*
- void [launchPeriodicTasks \(\)](#)  
*Main scheduler function.*
- void [startScheduling \(\)](#)  
*Starts the tasks scheduling.*
- void [addPeriodicTask \(TaskPtr\\_t task\\_ptr, uint16\\_t a\\_period\)](#)  
*Add a task into the scheduler.*
- bool [removePeriodicTask \(TaskPtr\\_t task\\_ptr\)](#)  
*Remove a task from the scheduler.*
- uint32\_t [getPitNumber \(\)](#)  
*Get function for PIT number.*
- bool [updateTaskPeriod \(TaskPtr\\_t task\\_ptr, uint16\\_t period\)](#)  
*Task period update function.*
- uint8\_t [getTaskCount \(\)](#)  
*Task count get function.*

## Static Public Member Functions

- static bool [LLElementCompare \(void \\*LLElement, void \\*CompareElement\)](#)  
*Linked list comparison function.*

## Private Types

- typedef struct [scheduler::Task\\_t Task\\_t](#)  
*Type defining a task structure.*

## Private Attributes

- uint8\_t [task\\_count](#)
- [LinkedList \\* TasksLL\\_ptr](#)
- uint32\_t [pit\\_number](#)

### 3.13.1 Detailed Description

Scheduler class.

This class defines the scheduler of the system.

It is called by the main interrupt and calls successively all applicative functions according to their recurrence time.  
All tasks called by the scheduler shall have the following prototype : static void task();

Definition at line 30 of file scheduler.h.

### 3.13.2 Member Typedef Documentation

#### 3.13.2.1 Task\_t

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

Type defining a task structure.

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

### 3.13.3 Constructor & Destructor Documentation

#### 3.13.3.1 scheduler()

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

scheduler class constructor

This function initializes the class scheduler

Returns

Nothing

Definition at line 29 of file scheduler.cpp.

Here is the call graph for this function:



### 3.13.4 Member Function Documentation

#### 3.13.4.1 addPeriodicTask()

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

Add a task into the scheduler.

This function create a new task in the scheduler linked to the function task\_ptr with a period a\_period and an ID a\_task\_id

**Parameters**

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

**Returns**

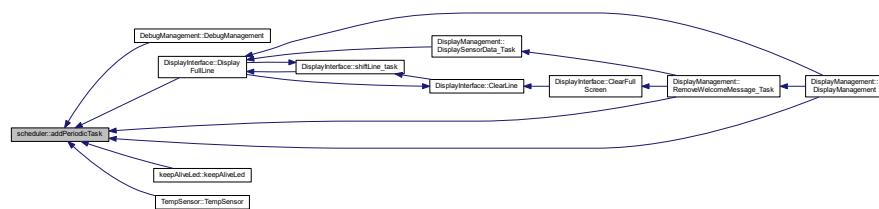
Nothing

Definition at line 99 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.13.4.2 getPitNumber()**

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

Get function for PIT number.

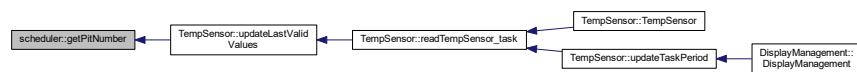
This function returns the PIT number

**Returns**

PIT number

Definition at line 113 of file scheduler.cpp.

Here is the caller graph for this function:



#### 3.13.4.3 getTaskCount()

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

Task count get function.

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

##### Returns

Number of tasks

Definition at line 115 of file scheduler.h.

#### 3.13.4.4 launchPeriodicTasks()

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

Main scheduler function.

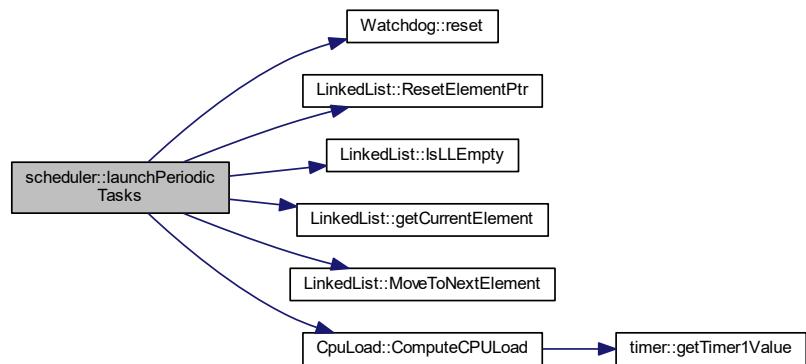
This function launches the scheduled tasks according to current software time and task configuration

##### Returns

Nothing

Definition at line 54 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.13.4.5 LLElementCompare()

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

Linked list comparison function.

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

#### Parameters

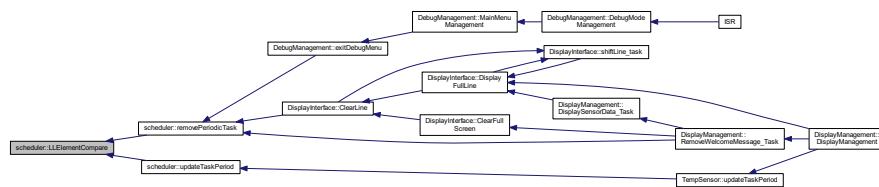
in	<i>LLElement</i>	Pointer to the linked list element
in	<i>CompareElement</i>	Pointer to the element to the compare

#### Returns

True if both elements are identical, false otherwise

Definition at line 131 of file scheduler.cpp.

Here is the caller graph for this function:



### 3.13.4.6 removePeriodicTask()

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

Remove a task from the scheduler.

This function finds the task defined by *task\_ptr* in the scheduler and removes it.

#### Parameters

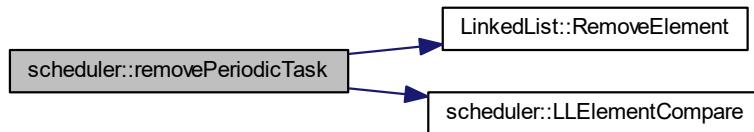
in	<i>task_ptr</i>	address of the task to remove from scheduler
----	-----------------	--

**Returns**

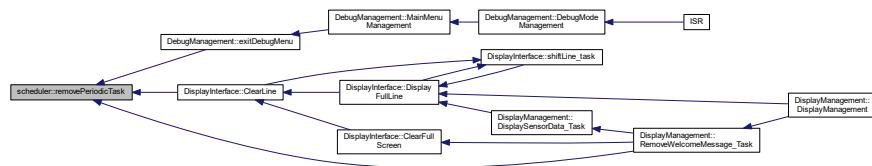
TRUE if the task has been removed, FALSE if the task does not exist in the scheduler

Definition at line 119 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.13.4.7 startScheduling()**

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

Starts the tasks scheduling.

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

**Returns**

Nothing

Definition at line 93 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.13.4.8 updateTaskPeriod()

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

Task period update function.

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

#### Parameters

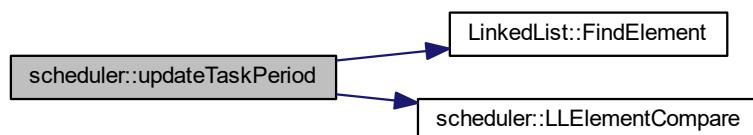
in	<i>task_ptr</i>	Pointer of the task to update
in	<i>period</i>	New period of the task

#### Returns

True if the update has been correctly done, false otherwise

Definition at line 142 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.13.5 Member Data Documentation

#### 3.13.5.1 pit\_number

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

Counter of periodic interrupts

Definition at line 140 of file scheduler.h.

#### 3.13.5.2 task\_count

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

Number of task in scheduler

Definition at line 136 of file scheduler.h.

#### 3.13.5.3 TasksLL\_ptr

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

Pointer to the linked list object containing the tasks

Definition at line 138 of file scheduler.h.

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

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

## 3.14 String Class Reference

`String` management class.

```
#include <String.h>
```

### Public Member Functions

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

### Private Member Functions

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

### Private Attributes

- `uint8_t * string`
- `uint8_t size`

#### 3.14.1 Detailed Description

`String` management class.

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

Definition at line 18 of file String.h.

### 3.14.2 Constructor & Destructor Documentation

#### 3.14.2.1 String() [1/2]

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

Class constructor.

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

##### Parameters

in	str	Pointer to initialization string
----	-----	----------------------------------

##### Returns

Nothing

Definition at line 15 of file String.cpp.

Here is the call graph for this function:



#### 3.14.2.2 String() [2/2]

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

Class constructor.

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

##### Returns

Nothing

Definition at line 33 of file String.cpp.

### 3.14.2.3 ~String()

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

Class destructor.

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

#### Returns

Nothing

Definition at line 39 of file String.cpp.

Here is the call graph for this function:



## 3.14.3 Member Function Documentation

### 3.14.3.1 appendBool()

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

Boolean adding function.

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

#### Parameters

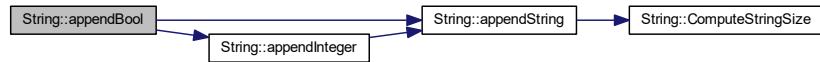
in	<i>data</i>	Boolean data to add
in	<i>isText</i>	Defines the conversion mode : text or integer

#### Returns

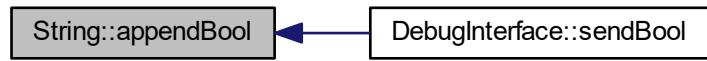
Nothing

Definition at line 121 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.14.3.2 appendChar()

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

Character adding function.

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

#### Parameters

in	<i>data</i>	1-byte character to add
----	-------------	-------------------------

#### Returns

Nothing

Definition at line 139 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.14.3.3 appendInteger()

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

Integer adding function.

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

#### Parameters

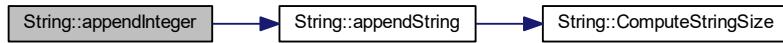
in	<b>value</b>	Integer to add
in	<b>base</b>	Base of computation of the integer (between 2 and 36)

#### Returns

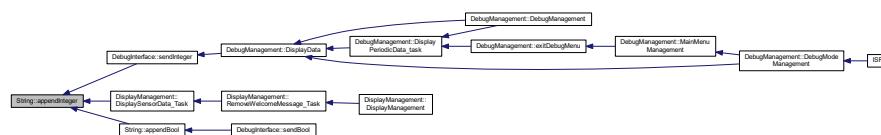
Nothing

Definition at line 95 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 3.14.3.4 appendString()

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

[String](#) adding function.

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

#### Parameters

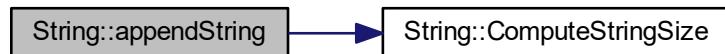
in	str	New string to add
----	-----	-------------------

#### Returns

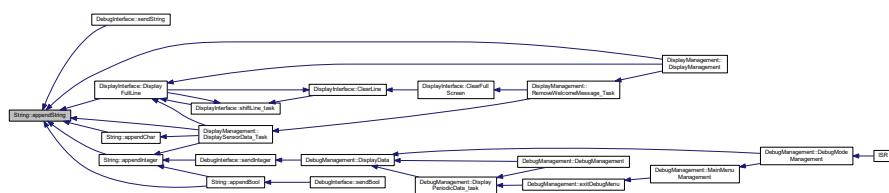
Nothing

Definition at line 57 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 3.14.3.5 Clear()

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

[String](#) clear function.

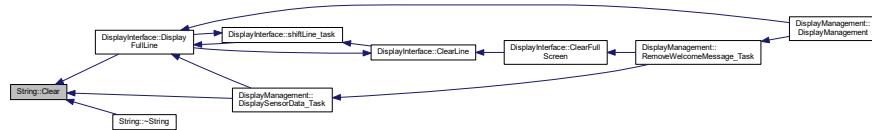
This function clears the string. Size is set to 0 and the memory is freed.

**Returns**

Nothing

Definition at line 113 of file String.cpp.

Here is the caller graph for this function:

**3.14.3.6 ComputeStringSize()**

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

[String](#) size computation function.

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

**Parameters**

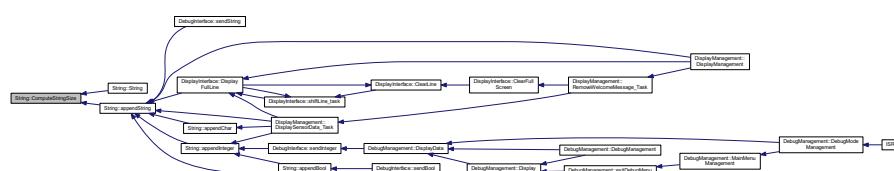
in	<i>str</i>	Pointer to the beginning of the string
----	------------	--

**Returns**

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

Definition at line 44 of file String.cpp.

Here is the caller graph for this function:



### 3.14.3.7 getSize()

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

Size get function.

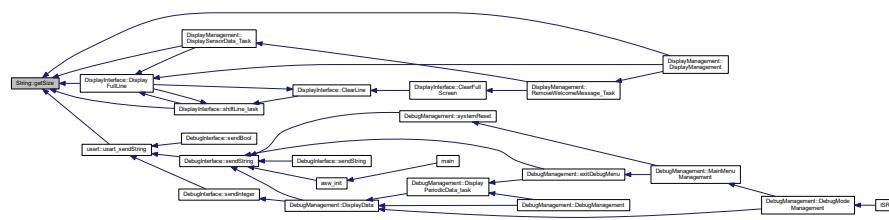
This function returns the size of the string.

#### Returns

Size of the string

Definition at line 64 of file String.h.

Here is the caller graph for this function:



### 3.14.3.8 getString()

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

`String` pointer get function.

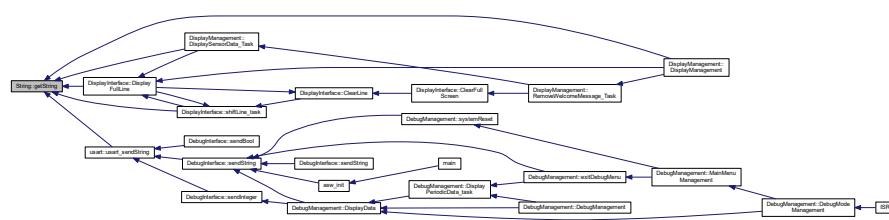
This function returns the pointer to the beginning of the string.

#### Returns

`String` pointer

Definition at line 53 of file String.h.

Here is the caller graph for this function:



### 3.14.4 Member Data Documentation

#### 3.14.4.1 size

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

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

Definition at line 121 of file String.h.

#### 3.14.4.2 string

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

Pointer to the start of the string

Definition at line 120 of file String.h.

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

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

## 3.15 T\_ASW\_init\_cnf Struct Reference

ASW initialization configuration structure.

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

### Public Attributes

- bool [isDebugEnabled](#)
- bool [isLEDACTivated](#)
- bool [isTempSensorACTivated](#)
- bool [isDisplayACTivated](#)

#### 3.15.1 Detailed Description

ASW initialization configuration structure.

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

Definition at line 17 of file asw.h.

### 3.15.2 Member Data Documentation

#### 3.15.2.1 isDebugEnabled

```
bool T_ASW_init_cnf::isDebugEnabled
```

Debug services activation flag

Definition at line 19 of file asw.h.

#### 3.15.2.2 isDisplayActivated

```
bool T_ASW_init_cnf::isDisplayActivated
```

LCD display activation flag

Definition at line 22 of file asw.h.

#### 3.15.2.3 isLEDActivated

```
bool T_ASW_init_cnf::isLEDActivated
```

Keep-alive LED activation flag

Definition at line 20 of file asw.h.

#### 3.15.2.4 isTempSensorActivated

```
bool T_ASW_init_cnf::isTempSensorActivated
```

Temperature sensor activation flag

Definition at line 21 of file asw.h.

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

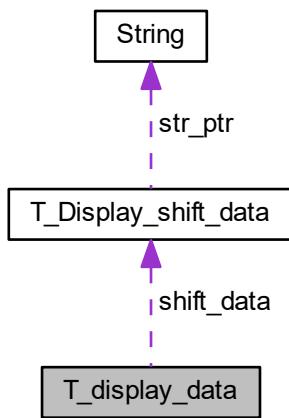
- [asw.h](#)

### 3.16 T\_display\_data Struct Reference

Structure containing display data.

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

Collaboration diagram for T\_display\_data:



#### Public Attributes

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

#### 3.16.1 Detailed Description

Structure containing display data.

This structure contains all data used for screen display

Definition at line 57 of file `DisplayInterface.h`.

#### 3.16.2 Member Data Documentation

### 3.16.2.1 alignment

`T_DisplayInterface_LineAlignment` `T_display_data::alignment`

Line alignment

Definition at line 61 of file DisplayInterface.h.

### 3.16.2.2 display\_str

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

Current string displayed on the screen

Definition at line 63 of file DisplayInterface.h.

### 3.16.2.3 isEmpty

`bool` `T_display_data::isEmpty`

Flag indicating if the line is empty or not

Definition at line 59 of file DisplayInterface.h.

### 3.16.2.4 mode

`T_DisplayInterface_LineDisplayMode` `T_display_data::mode`

Current display mode

Definition at line 60 of file DisplayInterface.h.

### 3.16.2.5 shift\_data

`T_Display_shift_data` `T_display_data::shift_data`

Shift data for the current line

Definition at line 62 of file DisplayInterface.h.

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

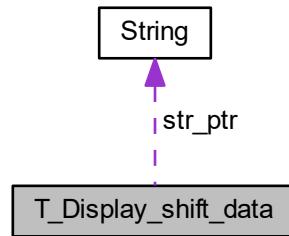
- [DisplayInterface.h](#)

### 3.17 T\_Display\_shift\_data Struct Reference

Structure containing shift data.

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

Collaboration diagram for T\_Display\_shift\_data:



#### Public Attributes

- `String * str_ptr`
- `uint8_t * str_cur_ptr`
- `uint8_t temporization`

#### 3.17.1 Detailed Description

Structure containing shift data.

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

Definition at line 45 of file DisplayInterface.h.

#### 3.17.2 Member Data Documentation

##### 3.17.2.1 str\_cur\_ptr

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

Pointer to the address of the first displayed character

Definition at line 48 of file DisplayInterface.h.

### 3.17.2.2 str\_ptr

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

Pointer to the start address of the string

Definition at line 47 of file DisplayInterface.h.

### 3.17.2.3 temporization

```
uint8_t T_Display_shift_data::temporization
```

Shifting period

Definition at line 49 of file DisplayInterface.h.

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

- [DisplayInterface.h](#)

## 3.18 T\_LCD\_conf\_struct Struct Reference

Structure defining [LCD](#) configuration.

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

### Public Attributes

- uint32\_t [i2c\\_bitrate](#)
- uint8\_t [i2c\\_addr](#)
- bool [backlight\\_en](#)
- bool [lineNumber\\_cnf](#)
- bool [fontType\\_cnf](#)
- bool [display\\_en](#)
- bool [cursor\\_en](#)
- bool [cursorBlink\\_en](#)
- bool [entryModeDir](#)
- bool [entryModeShift](#)

### 3.18.1 Detailed Description

Structure defining [LCD](#) configuration.

Definition at line 128 of file LCD.h.

### 3.18.2 Member Data Documentation

#### 3.18.2.1 `backlight_en`

```
bool T_LCD_conf_struct::backlight_en
```

Screen backlight enable flag

Definition at line 132 of file LCD.h.

#### 3.18.2.2 `cursor_en`

```
bool T_LCD_conf_struct::cursor_en
```

Screen cursor enable flag

Definition at line 136 of file LCD.h.

#### 3.18.2.3 `cursorBlink_en`

```
bool T_LCD_conf_struct::cursorBlink_en
```

Screen cursor blinking enable flag

Definition at line 137 of file LCD.h.

#### 3.18.2.4 `display_en`

```
bool T_LCD_conf_struct::display_en
```

Screen display enable flag

Definition at line 135 of file LCD.h.

#### 3.18.2.5 `entryModeDir`

```
bool T_LCD_conf_struct::entryModeDir
```

Entry mode direction configuration

Definition at line 138 of file LCD.h.

### 3.18.2.6 entryModeShift

```
bool T_LCD_conf_struct::entryModeShift
```

Entry mode shift configuration

Definition at line 139 of file LCD.h.

### 3.18.2.7 fontType\_cnf

```
bool T_LCD_conf_struct::fontType_cnf
```

Font configuration

Definition at line 134 of file LCD.h.

### 3.18.2.8 i2c\_addr

```
uint8_t T_LCD_conf_struct::i2c_addr
```

I<sup>2</sup>C address if the screen

Definition at line 131 of file LCD.h.

### 3.18.2.9 i2c\_bitrate

```
uint32_t T_LCD_conf_struct::i2c_bitrate
```

I<sup>2</sup>C bitrate needed by the LCD screen

Definition at line 130 of file LCD.h.

### 3.18.2.10 lineNumber\_cnf

```
bool T_LCD_conf_struct::lineNumber_cnf
```

Screen line number configuration (1 or 2 lines)

Definition at line 133 of file LCD.h.

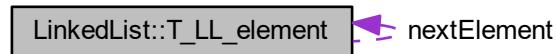
The documentation for this struct was generated from the following file:

- [LCD.h](#)

### 3.19 LinkedList::T\_LL\_element Struct Reference

Type defining a linked list element.

Collaboration diagram for LinkedList::T\_LL\_element:



#### Public Attributes

- void \* [data\\_ptr](#)
- [T\\_LL\\_element](#) \* [nextElement](#)

#### 3.19.1 Detailed Description

Type defining a linked list element.

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

Definition at line 117 of file [LinkedList.h](#).

#### 3.19.2 Member Data Documentation

##### 3.19.2.1 [data\\_ptr](#)

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

Definition at line 119 of file [LinkedList.h](#).

##### 3.19.2.2 [nextElement](#)

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

Definition at line 120 of file [LinkedList.h](#).

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

- [LinkedList.h](#)

## 3.20 scheduler::Task\_t Struct Reference

Type defining a task structure.

### Public Attributes

- [TaskPtr\\_t TaskPtr](#)
- [uint16\\_t period](#)

#### 3.20.1 Detailed Description

Type defining a task structure.

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

Definition at line 129 of file scheduler.h.

#### 3.20.2 Member Data Documentation

##### 3.20.2.1 period

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

Period of the task

Definition at line 132 of file scheduler.h.

##### 3.20.2.2 TaskPtr

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

Pointer to the task

Definition at line 131 of file scheduler.h.

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

- [scheduler.h](#)

## 3.21 TempSensor Class Reference

Class for temperature sensor.

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

## Public Member Functions

- **TempSensor ()**  
*Class constructor.*
- **uint16\_t \* getTempPtr ()**  
*Get pointer to data raw\_temperature.*
- **uint16\_t \* getHumPtr ()**  
*Get pointer to data raw\_humidity.*
- **bool getTemp (uint16\_t \*temp)**  
*Get temperature data.*
- **bool getHumidity (uint16\_t \*hum)**  
*Get humidity data.*
- **void setValidity (bool validity)**  
*Set data val\_validity.*
- **void updateLastValidValues ()**
- **uint8\_t GetTempInteger ()**  
*Temperature formatting function - Integer part.*
- **uint8\_t GetTempDecimal ()**  
*Temperature formatting function - Decimal part.*
- **uint8\_t GetHumInteger ()**  
*Humidity formatting function - Integer part.*
- **uint8\_t GetHumDecimal ()**  
*Humidity formatting function - Decimal part.*
- **bool GetValidity ()**  
*Data validity get function.*
- **bool updateTaskPeriod (uint16\_t period)**  
*Task period update.*
- **uint16\_t getTaskPeriod ()**  
*Task period get function.*

## Static Public Member Functions

- **static void readTempSensor\_task ()**  
*Task for reading temperature and humidity values.*

## Private Attributes

- **uint16\_t read\_temperature**
- **uint16\_t read\_humidity**
- **bool validity\_last\_read**
- **bool validity**
- **uint32\_t valid\_pit**
- **uint16\_t valid\_temp**
- **uint16\_t valid\_hum**
- **uint16\_t task\_period**

### 3.21.1 Detailed Description

Class for temperature sensor.

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

Definition at line 21 of file TempSensor.h.

### 3.21.2 Constructor & Destructor Documentation

#### 3.21.2.1 TempSensor()

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

Class constructor.

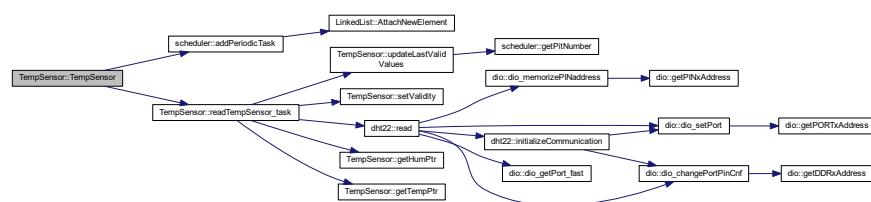
This function initializes all data of the class [TempSensor](#). If needed, it creates a new instance of the DHT22 sensor object. It also adds the periodic task in the scheduler.

Returns

Nothing

Definition at line 24 of file [TempSensor.cpp](#).

Here is the call graph for this function:



### 3.21.3 Member Function Documentation

#### 3.21.3.1 GetHumDecimal()

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

Humidity formatting function - Decimal part.

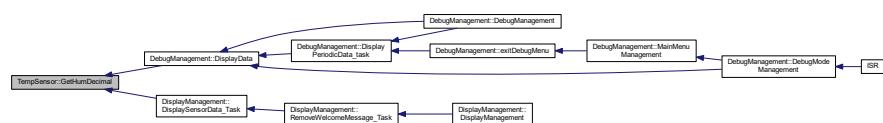
This function return the decimal part of the humidity

Returns

Decimal value of the humidity

Definition at line 124 of file [TempSensor.h](#).

Here is the caller graph for this function:



### 3.21.3.2 getHumidity()

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

Get humidity data.

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

#### Parameters

out	hum	Humidity value
-----	-----	----------------

#### Returns

Validity of humidity

Definition at line 80 of file TempSensor.cpp.

### 3.21.3.3 GetHumInteger()

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

Humidity formatting function - Integer part.

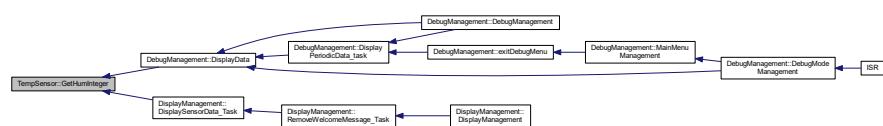
This function return the integer part of the humidity

#### Returns

Integer value of the humidity

Definition at line 113 of file TempSensor.h.

Here is the caller graph for this function:



### 3.21.3.4 getHumPtr()

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

Get pointer to data raw\_humidity.

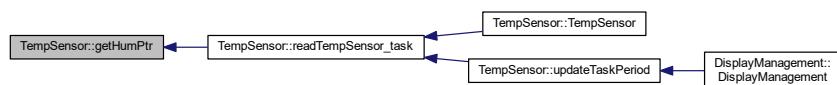
This function returns a pointer to the class member raw\_humidity

#### Returns

Pointer to raw\_humidity

Definition at line 55 of file TempSensor.cpp.

Here is the caller graph for this function:



### 3.21.3.5 getTaskPeriod()

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

Task period get function.

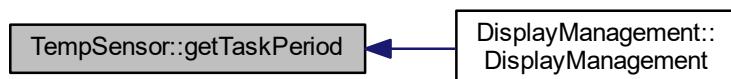
This function returns the period of the sensor task

#### Returns

Period of the task (ms)

Definition at line 155 of file TempSensor.h.

Here is the caller graph for this function:



### 3.21.3.6 getTemp()

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

Get temperature data.

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

**Parameters**

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

**Returns**

Validity of temperature

Definition at line 86 of file TempSensor.cpp.

**3.21.3.7 GetTempDecimal()**

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

Temperature formatting function - Decimal part.

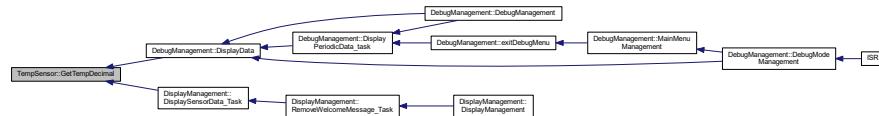
This function return the decimal part of the temperature

**Returns**

Decimal value of the temperature

Definition at line 102 of file TempSensor.h.

Here is the caller graph for this function:

**3.21.3.8 GetTempInteger()**

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

Temperature formatting function - Integer part.

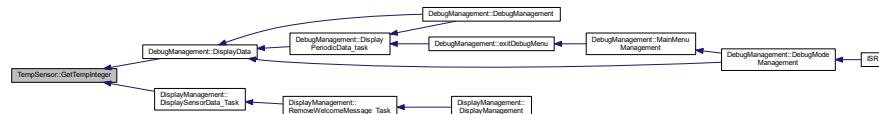
This function return the integer part of the temperature

**Returns**

Integer value of the temperature

Definition at line 91 of file TempSensor.h.

Here is the caller graph for this function:



### 3.21.3.9 getTempPtr()

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

Get pointer to data raw\_temperature.

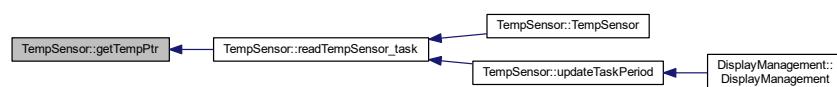
This function returns a pointer to the class member raw\_temperature

#### Returns

Pointer to raw\_temperature

Definition at line 60 of file TempSensor.cpp.

Here is the caller graph for this function:



### 3.21.3.10 GetValidity()

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

Data validity get function.

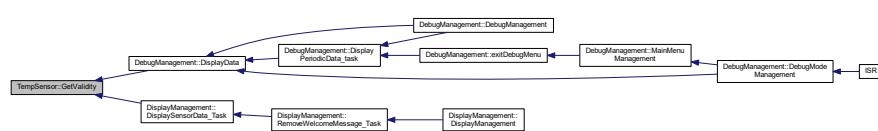
This function returns the validity of the sensor data

#### Returns

True if the sensor values are valid, false otherwise

Definition at line 135 of file TempSensor.h.

Here is the caller graph for this function:



### 3.21.3.11 `readTempSensor_task()`

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

Task for reading temperature and humidity values.

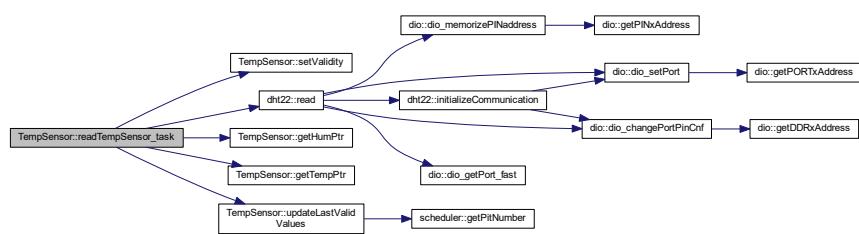
This task reads temperature and humidity data using DHT22 driver. It is called every 5 seconds.

#### Returns

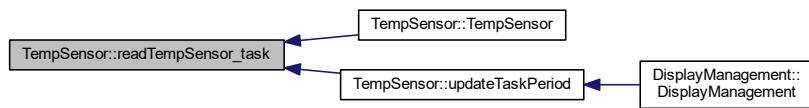
Nothing

Definition at line 44 of file TempSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.21.3.12 `setValidity()`

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

Set data `val_validity`.

This function sets the class member `val_validity`

#### Parameters

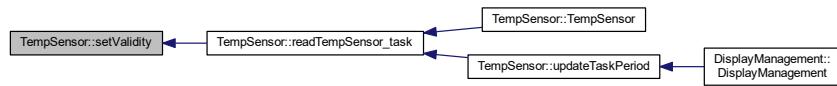
in	<code>validity</code>	Value of validity
----	-----------------------	-------------------

Returns

Nothing

Definition at line 50 of file TempSensor.cpp.

Here is the caller graph for this function:

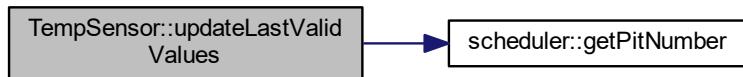


### 3.21.3.13 updateLastValidValues()

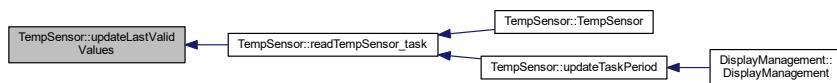
```
void TempSensor::updateLastValidValues ( )
```

Definition at line 65 of file TempSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.21.3.14 updateTaskPeriod()

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

Task period update.

This function updates the period of the temperature task.

**Parameters**

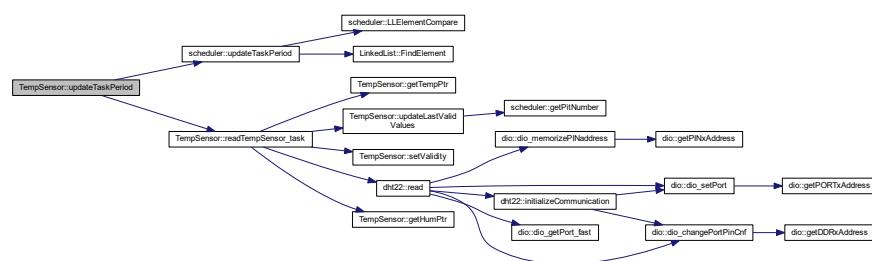
in	<i>period</i>	New period of the task
----	---------------	------------------------

**Returns**

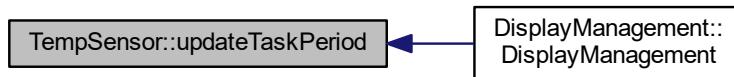
True if the period has been updated, false otherwise

Definition at line 92 of file TempSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.21.4 Member Data Documentation

#### 3.21.4.1 read\_humidity

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

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

Definition at line 163 of file TempSensor.h.

#### 3.21.4.2 `read_temperature`

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

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

Definition at line 162 of file TempSensor.h.

#### 3.21.4.3 `task_period`

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

Task period

Definition at line 172 of file TempSensor.h.

#### 3.21.4.4 `valid_hum`

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

Valid value of humidity

Definition at line 170 of file TempSensor.h.

#### 3.21.4.5 `valid坑`

```
uint32_t TempSensor::valid坑 [private]
```

pit number of the last time when data were valid

Definition at line 167 of file TempSensor.h.

#### 3.21.4.6 `valid_temp`

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

Valid value of temperature

Definition at line 169 of file TempSensor.h.

### 3.21.4.7 validity

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

validity of official temperature and humidity data

Definition at line 166 of file TempSensor.h.

### 3.21.4.8 validity\_last\_read

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

Validity of last read temperature and humidity data

Definition at line 164 of file TempSensor.h.

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

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

## 3.22 timer Class Reference

Class defining a timer.

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

### Public Member Functions

- [timer \(\)](#)  
*Class constructor.*
- [void configureTimer1 \(uint16\\_t a\\_prescaler, uint16\\_t a\\_ctcValue\)](#)  
*Configures Timer #1.*
- [void startTimer1 \(\)](#)  
*Start Timer #1.*
- [void stopTimer1 \(\)](#)  
*Stops Timer #1.*
- [uint16\\_t getTimer1Value \(\)](#)  
*Reads current value of timer #1.*

### Private Attributes

- [uint8\\_t prescaler](#)

### 3.22.1 Detailed Description

Class defining a timer.

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

Definition at line 19 of file timer.h.

### 3.22.2 Constructor & Destructor Documentation

#### 3.22.2.1 timer()

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

Class constructor.

This function initializes class attributes

Returns

Nothing

Definition at line 15 of file timer.cpp.

### 3.22.3 Member Function Documentation

#### 3.22.3.1 configureTimer1()

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

Configures Timer #1.

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

Parameters

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

**Returns**

Nothing

Definition at line 20 of file timer.cpp.

Here is the caller graph for this function:



### 3.22.3.2 getTimer1Value()

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

Reads current value of timer #1.

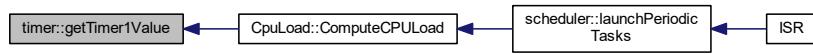
This function reads the value of of timer #1 using register TCNT1. The function is inlined to speed up SW execution.

**Returns**

Current timer value

Definition at line 58 of file timer.h.

Here is the caller graph for this function:



### 3.22.3.3 startTimer1()

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

Start Timer #1.

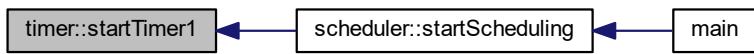
This functions starts Timer #1. Timer shall be initialized before this function is called.

#### Returns

Nothing

Definition at line 58 of file timer.cpp.

Here is the caller graph for this function:



### 3.22.3.4 stopTimer1()

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

Stops Timer #1.

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

#### Returns

Nothing

Definition at line 69 of file timer.cpp.

## 3.22.4 Member Data Documentation

### 3.22.4.1 prescaler

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

Definition at line 64 of file timer.h.

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

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

## 3.23 usart Class Reference

USART serial bus class.

```
#include <uart.h>
```

### Public Member Functions

- **uart** (uint16\_t a\_BaudRate)  
*Class usart constructor.*
- void **uart\_sendString** (String \*str)  
*Send a string on USART link.*
- void **uart\_sendByte** (uint8\_t data)  
*Send a single byte on USART link.*
- void **setBaudRate** (uint16\_t a\_BaudRate)  
*Setting baud rate.*
- void **uart\_init** ()  
*USART hardware initialization.*
- uint8\_t **uart\_read** ()  
*USART read function.*

### Private Member Functions

- void **uart\_transmit** (uint8\_t Data)  
*USART Transmit data.*

### Private Attributes

- uint16\_t **BaudRate**

#### 3.23.1 Detailed Description

USART serial bus class.

This class defines all useful functions for USART serial bus

Definition at line 16 of file usart.h.

#### 3.23.2 Constructor & Destructor Documentation

##### 3.23.2.1 usart()

```
uart::uart (
    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 18 of file usart.cpp.

Here is the call graph for this function:



### 3.23.3 Member Function Documentation

#### 3.23.3.1 setBaudRate()

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

Setting baud rate.

This function sets the attribute BaudRate of the class usart

**Parameters**

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

**Returns**

Nothing

Definition at line 74 of file usart.cpp.

### 3.23.3.2 usart\_init()

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

USART hardware initialization.

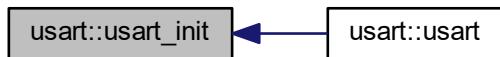
This function will initialize the USART using selected baudrate. User must pay attention to select one of the usually used Baud Rate (9600, 19200, 38400, 57600). Note that since an uint16 is used as argument, Baud rate cannot be more than 57600.

#### Returns

Nothing.

Definition at line 25 of file usart.cpp.

Here is the caller graph for this function:



### 3.23.3.3 usart\_read()

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

USART read function.

This function will read reception register of USART

#### Returns

The function returns the 8 bits read from reception buffer

Definition at line 90 of file usart.cpp.

### 3.23.3.4 usart\_sendByte()

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

Send a single byte on USART link.

This function writes the given byte to the serial link using usart\_transmit function

**Parameters**

in	<i>data</i>	Data byte being sent
----	-------------	----------------------

**Returns**

Nothing.

Definition at line 68 of file usart.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.23.3.5 usart\_sendString()**

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

Send a string on USART link.

This function writes the string object data to the serial link using usart\_transmit function

**Parameters**

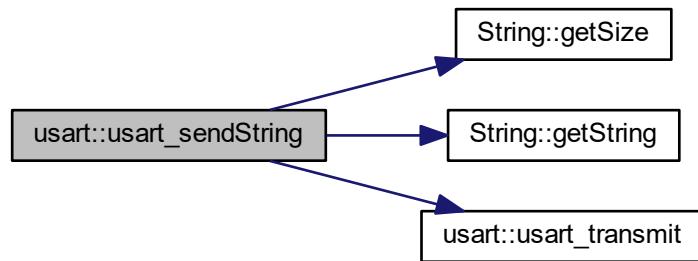
in	<i>str</i>	Pointer to the string being sent
----	------------	----------------------------------

**Returns**

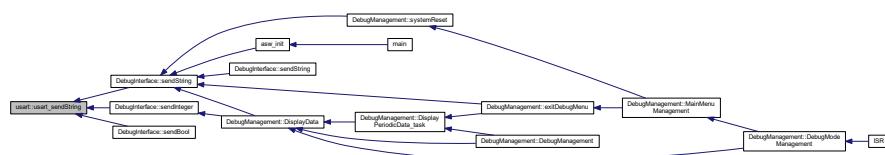
Nothing.

Definition at line 48 of file usart.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.23.3.6 usart\_transmit()

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

USART Transmit data.

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

#### Parameters

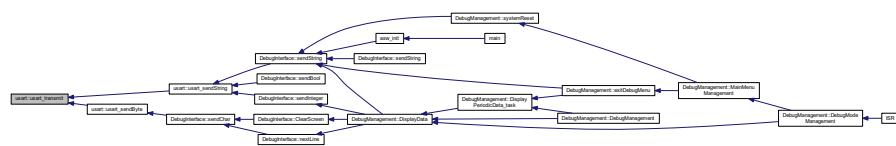
in	<i>Data</i>	Desired data char to transmit
----	-------------	-------------------------------

#### Returns

Nothing.

Definition at line 81 of file `usart.cpp`.

Here is the caller graph for this function:



### **3.23.4 Member Data Documentation**

### 3.23.4.1 BaudRate

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

Defines the baud rate used by driver

Definition at line 77 of file usart.h.

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

- `uart.h`
  - `uart.cpp`

## 3.24 Watchdog Class Reference

## [Watchdog](#) management class.

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

## Public Member Functions

- **Watchdog ()**  
*Class constructor.*
  - **Watchdog (uint8\_t timeout)**  
*Overloaded class constructor.*
  - void **reset ()**  
*Watchdog reset function.*
  - void **timeoutUpdate (uint8\_t value)**  
*Watchdog timeout value update function.*
  - void **SystemReset ()**  
*System reset function.*

## Private Member Functions

- void `enable` (uint8\_t value)  
*Watchdog enabling function.*
- void `disable` ()  
*Watchdog disabling function.*

### 3.24.1 Detailed Description

`Watchdog` management class.

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

Definition at line 31 of file Watchdog.h.

### 3.24.2 Constructor & Destructor Documentation

#### 3.24.2.1 `Watchdog()` [1/2]

`Watchdog::Watchdog ( )`

Class constructor.

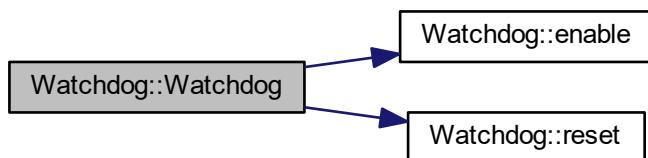
This function initializes the watchdog class. It enables the HW watchdog with a default timeout value.

Returns

Nothing

Definition at line 22 of file Watchdog.cpp.

Here is the call graph for this function:



#### 3.24.2.2 `Watchdog()` [2/2]

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

Overloaded class constructor.

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

**Parameters**

in	<i>timeout</i>	Timeout value requested for the watchdog
----	----------------	--

**Returns**

Nothing

Definition at line 28 of file Watchdog.cpp.

Here is the call graph for this function:

**3.24.3 Member Function Documentation****3.24.3.1 disable()**

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

[Watchdog](#) disabling function.

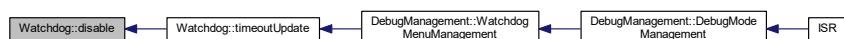
This function disables the watchdog by calling `wdt_disable` macro.

**Returns**

Nothing

Definition at line 39 of file Watchdog.cpp.

Here is the caller graph for this function:

**3.24.3.2 enable()**

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

[Watchdog](#) enabling function.

This function enables the watchdog by calling `wdt_enable` macro.

**Parameters**

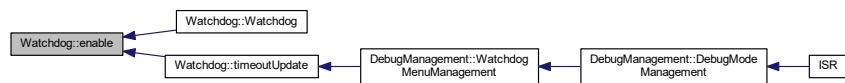
in	<i>value</i>	Timeout value
----	--------------	---------------

**Returns**

Nothing

Definition at line 34 of file Watchdog.cpp.

Here is the caller graph for this function:

**3.24.3.3 reset()**

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

[Watchdog](#) reset function.

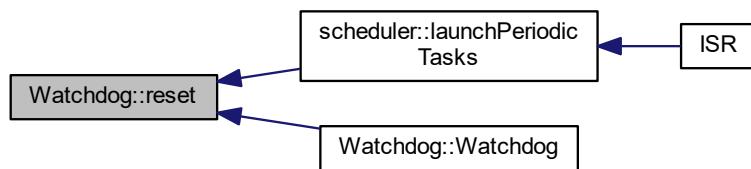
This function resets the watchdog timer by calling `wdt_reset` macro

**Returns**

Nothing

Definition at line 44 of file Watchdog.cpp.

Here is the caller graph for this function:



#### 3.24.3.4 SystemReset()

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

System reset function.

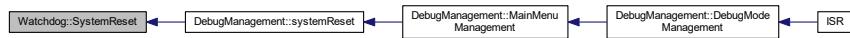
This function provokes a system reset by going in an infinite loop. Thus the watchdog will reset the CPU when the timeout occurs.

##### Returns

Nothing

Definition at line 56 of file Watchdog.cpp.

Here is the caller graph for this function:



#### 3.24.3.5 timeoutUpdate()

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

[Watchdog](#) timeout value update function.

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

##### Parameters

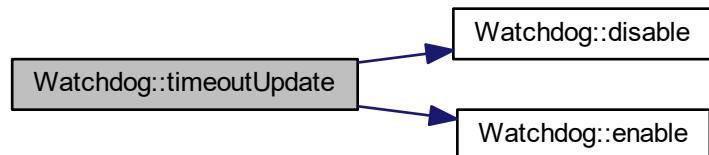
in	value	New timeout value
----	-------	-------------------

**Returns**

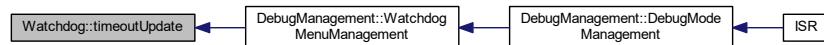
Nothing

Definition at line 49 of file Watchdog.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:

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

## Chapter 4

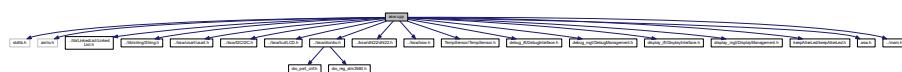
# File Documentation

### 4.1 asw.cpp File Reference

ASW main file.

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

Include dependency graph for asw.cpp:



### Functions

- void **asw\_init ()**

*Initialization of ASW.*

### 4.1.1 Detailed Description

ASW main file.

#### Date

15 mars 2018

#### Author

nicls67

### 4.1.2 Function Documentation

#### 4.1.2.1 asw\_init()

```
void asw_init ( )
```

Initialization of ASW.

This function instantiates all applicative objects. Some objects are not created by this function but directly by the upper-level class. The addresses of objects are then stored in ASW\_cnf\_struct structure.

The debug interface object is created only if the debug pin is set to logical high level.

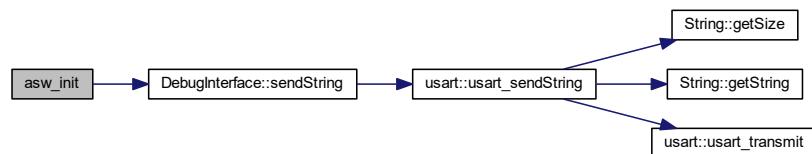
This function shall be called after BSW initialization function.

#### Returns

Nothing

Definition at line 37 of file asw.cpp.

Here is the call graph for this function:



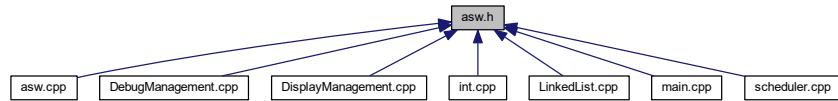
Here is the caller graph for this function:



## 4.2 asw.h File Reference

ASW main header file.

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



### Classes

- struct [T\\_ASW\\_init\\_cnf](#)  
*ASW initialization configuration structure.*

### Functions

- void [asw\\_init](#) ()  
*Initialization of ASW.*

#### 4.2.1 Detailed Description

ASW main header file.

##### Date

15 mars 2018

##### Author

nicls67

#### 4.2.2 Function Documentation

#### 4.2.2.1 asw\_init()

```
void asw_init ( )
```

Initialization of ASW.

This function instantiates all applicative objects. Some objects are not created by this function but directly by the upper-level class. The addresses of objects are then stored in ASW\_cnf\_struct structure.

The debug interface object is created only if the debug pin is set to logical high level.

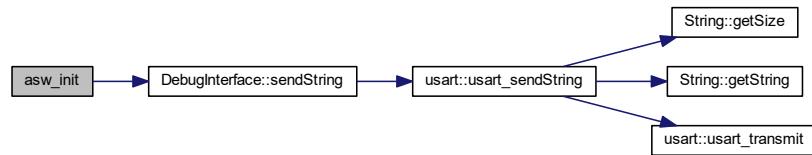
This function shall be called after BSW initialization function.

##### Returns

Nothing

Definition at line 37 of file asw.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



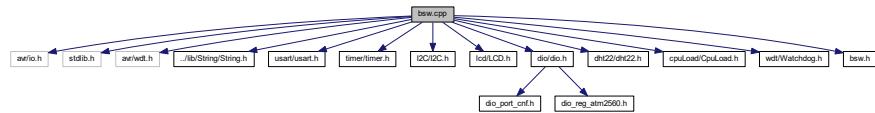
## 4.3 bsw.cpp File Reference

BSW main file.

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

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

Include dependency graph for bsw.cpp:
```



## Functions

- void [bsw\\_init \(\)](#)

*Initialization of BSW.*

### 4.3.1 Detailed Description

BSW main file.

Date

13 mars 2018

Author

nicls67

### 4.3.2 Function Documentation

#### 4.3.2.1 bsw\_init()

```
void bsw_init ( )
```

*Initialization of BSW.*

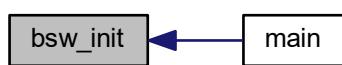
This function instantiates all driver objects, leading hardware initialization. The addresses of driver objects are then stored in BSW\_cnf\_struct structure.

Returns

Nothing

Definition at line 26 of file bsw.cpp.

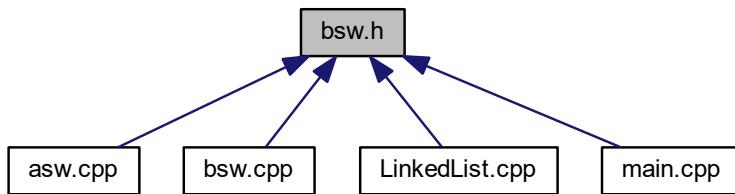
Here is the caller graph for this function:



## 4.4 bsw.h File Reference

BSW main header file.

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



### Functions

- void [bsw\\_init \(\)](#)

*Initialization of BSW.*

#### 4.4.1 Detailed Description

BSW main header file.

##### Date

13 mars 2018

##### Author

nicls67

#### 4.4.2 Function Documentation

#### 4.4.2.1 bsw\_init()

```
void bsw_init ( )
```

Initialization of BSW.

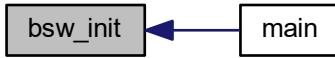
This function instantiates all driver objects, leading hardware initialization. The addresses of driver objects are then stored in BSW\_cnf\_struct structure.

#### Returns

Nothing

Definition at line 26 of file bsw.cpp.

Here is the caller graph for this function:

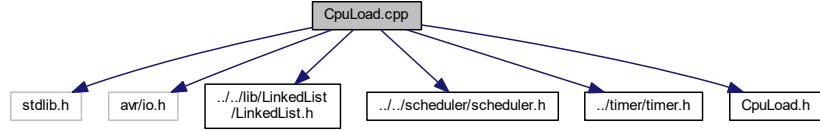


## 4.5 CpuLoad.cpp File Reference

Defines functions of class [CpuLoad](#).

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

Include dependency graph for CpuLoad.cpp:



## Variables

- [CpuLoad \\* p\\_global\\_BSW\\_cpuload](#)

#### 4.5.1 Detailed Description

Defines functions of class [CpuLoad](#).

##### Date

21 mars 2019

##### Author

nicls67

#### 4.5.2 Variable Documentation

##### 4.5.2.1 p\_global\_BSW\_cpuload

[CpuLoad](#)\* p\_global\_BSW\_cpuload

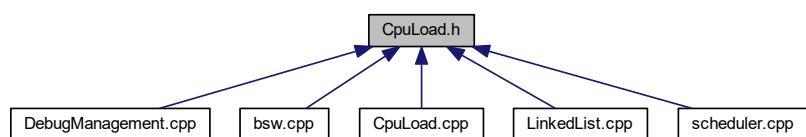
Pointer to cpu load library object

Definition at line 18 of file CpuLoad.cpp.

## 4.6 CpuLoad.h File Reference

[CpuLoad](#) class header file.

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



## Classes

- class [CpuLoad](#)

*Class defining CPU load libraries.*

## Macros

- #define [NB\\_OF\\_SAMPLES](#) 50

## Variables

- [CpuLoad \\* p\\_global\\_BSW\\_cpuload](#)

### 4.6.1 Detailed Description

[CpuLoad](#) class header file.

#### Date

21 mars 2019

#### Author

nicls67

### 4.6.2 Macro Definition Documentation

#### 4.6.2.1 NB\_OF\_SAMPLES

```
#define NB_OF_SAMPLES 50
```

Definition at line 13 of file [CpuLoad.h](#).

### 4.6.3 Variable Documentation

#### 4.6.3.1 p\_global\_BSW\_cpuload

```
CpuLoad* p_global_BSW_cpuload
```

Pointer to cpu load library object

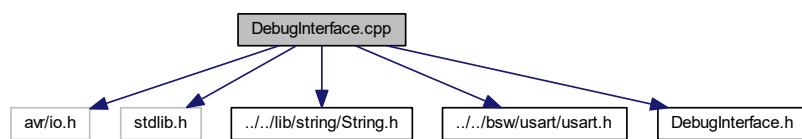
Definition at line 18 of file [CpuLoad.cpp](#).

## 4.7 DebugInterface.cpp File Reference

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

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

Include dependency graph for DebugInterface.cpp:



### Variables

- `DebugInterface * p_global_ASW_DebugInterface`

#### 4.7.1 Detailed Description

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

##### Date

15 mars 2018

##### Author

nicls67

#### 4.7.2 Variable Documentation

##### 4.7.2.1 `p_global_ASW_DebugInterface`

`DebugInterface* p_global_ASW_DebugInterface`

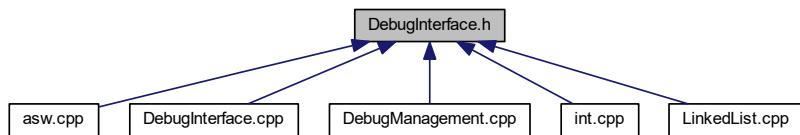
Pointer to USART debug interface object

Definition at line 19 of file DebugInterface.cpp.

## 4.8 DebugInterface.h File Reference

Header file for debug and logging functions.

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



### Classes

- class [DebugInterface](#)  
*Class used for debugging on usart link.*

### Macros

- #define [USART\\_BAUDRATE](#) (uint16\_t)9600

### Variables

- [DebugInterface \\* p\\_global\\_ASW\\_DebugInterface](#)

#### 4.8.1 Detailed Description

Header file for debug and logging functions.

##### Date

15 mars 2018

##### Author

nicls67

#### 4.8.2 Macro Definition Documentation

#### 4.8.2.1 USART\_BAUDRATE

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

uart connection to PC uses a baud rate of 9600

Definition at line 15 of file DebugInterface.h.

#### 4.8.3 Variable Documentation

##### 4.8.3.1 p\_global\_ASW\_DebugInterface

```
DebugInterface* p_global_ASW_DebugInterface
```

Pointer to USART debug interface object

Definition at line 19 of file DebugInterface.cpp.

### 4.9 DebugManagement.cpp File Reference

Debug management class source file.

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

Include dependency graph for DebugManagement.cpp:



## Variables

- `DebugManagement * p_global_ASW_DebugManagement`
- `const uint8_t str_debug_main_menu []`

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

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

*Watchdog timeout update selection.*
- `const uint8_t str_debug_info_message_empty [] = "\n"`

*Info menu empty string.*
- `const uint8_t str_debug_info_message_wrong_menu_selection [] = "Impossible de faire ca... !\n"`

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

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

### 4.9.1 Detailed Description

Debug management class source file.

#### Date

8 mai 2019

#### Author

nicls67

### 4.9.2 Variable Documentation

#### 4.9.2.1 p\_global\_ASW\_DebugManagement

`DebugManagement * p_global_ASW_DebugManagement`

Pointer to the `DebugManagement` object

Definition at line 30 of file `DebugManagement.cpp`.

#### 4.9.2.2 str\_debug\_info\_message\_empty

`const uint8_t str_debug_info_message_empty[] = "\n"`

Info menu empty string.

Definition at line 72 of file `DebugManagement.cpp`.

#### 4.9.2.3 str\_debug\_info\_message\_wdg\_tmo\_updated

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

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

Definition at line 82 of file DebugManagement.cpp.

#### 4.9.2.4 str\_debug\_info\_message\_wrong\_menu\_selection

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

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

Definition at line 77 of file DebugManagement.cpp.

#### 4.9.2.5 str\_debug\_main\_menu

```
const uint8_t str_debug_main_menu[ ]
```

##### **Initial value:**

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

Main menu of debug mode.

Definition at line 35 of file DebugManagement.cpp.

#### 4.9.2.6 str\_debug\_wdg\_menu

```
const uint8_t str_debug_wdg_menu[ ]
```

##### **Initial value:**

```
=
"Menu watchdog : \n"
"    1 : Changer timeout\n"
"\n"
"    q : Retour\n"
```

[Watchdog](#) menu of debug mode.

Definition at line 45 of file DebugManagement.cpp.

#### 4.9.2.7 str\_debug\_wdg\_timeout\_update\_selection

```
const uint8_t str_debug_wdg_timeout_update_selection[]
```

**Initial value:**

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

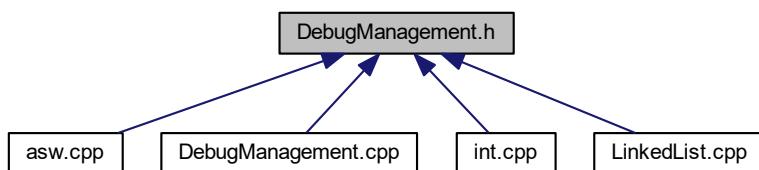
[Watchdog](#) timeout update selection.

Definition at line 54 of file DebugManagement.cpp.

## 4.10 DebugManagement.h File Reference

Debug management class header file.

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



### Classes

- struct [debug\\_mgt\\_state\\_struct\\_t](#)  
*Structure containing all debug states.*
- class [DebugManagement](#)  
*Debug management class.*

### Macros

- #define [PERIOD\\_MS\\_TASK\\_DISPLAY\\_DEBUG\\_DATA](#) 5000
- #define [PERIOD\\_MS\\_TASK\\_DISPLAY\\_CPU\\_LOAD](#) 5000

## Enumerations

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

## Variables

- `DebugManagement * p_global_ASW_DebugManagement`

### 4.10.1 Detailed Description

Debug management class header file.

#### Date

8 mai 2019

#### Author

nicls67

### 4.10.2 Macro Definition Documentation

#### 4.10.2.1 PERIOD\_MS\_TASK\_DISPLAY\_CPU\_LOAD

```
#define PERIOD_MS_TASK_DISPLAY_CPU_LOAD 5000
```

Period for displaying CPU load data

Definition at line 14 of file DebugManagement.h.

#### 4.10.2.2 PERIOD\_MS\_TASK\_DISPLAY\_DEBUG\_DATA

```
#define PERIOD_MS_TASK_DISPLAY_DEBUG_DATA 5000
```

Period for displaying temperature and humidity data

Definition at line 13 of file DebugManagement.h.

### 4.10.3 Enumeration Type Documentation

#### 4.10.3.1 debug\_mgt\_main\_menu\_state\_t

```
enum debug_mgt_main_menu_state_t
```

Defines the debug states.

**Enumerator**

MAIN_MENU	Init state : main menu is displayed
WDG_MENU	<a href="#">Watchdog</a> state : watchdog menu is displayed

Definition at line 20 of file DebugManagement.h.

**4.10.3.2 debug\_mgt\_wdg\_state\_t**

enum [debug\\_mgt\\_wdg\\_state\\_t](#)

Defines possible states for watchdog management.

**Enumerator**

WDG_MAIN	Main menu of watchdog management
WDG_TMO_UPDATE	Timeout update mode

Definition at line 30 of file DebugManagement.h.

**4.10.4 Variable Documentation****4.10.4.1 p\_global\_ASW\_DebugManagement**

[DebugManagement](#)\* p\_global\_ASW\_DebugManagement

Pointer to the [DebugManagement](#) object

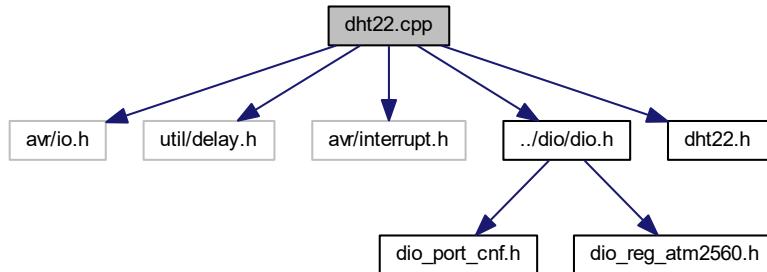
Definition at line 30 of file DebugManagement.cpp.

**4.11 dht22.cpp File Reference**

This file defines classes for DHT22 driver.

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

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



## Macros

- `#define MAX_WAIT_TIME_US 100`

## Variables

- `dht22 * p_global_BSW_dht22`

### 4.11.1 Detailed Description

This file defines classes for DHT22 driver.

#### Date

23 mars 2018

#### Author

nicls67

### 4.11.2 Macro Definition Documentation

#### 4.11.2.1 MAX\_WAIT\_TIME\_US

```
#define MAX_WAIT_TIME_US 100
```

Maximum waiting time in microseconds

Definition at line 18 of file dht22.cpp.

### 4.11.3 Variable Documentation

#### 4.11.3.1 p\_global\_BSW\_dht22

`dht22* p_global_BSW_dht22`

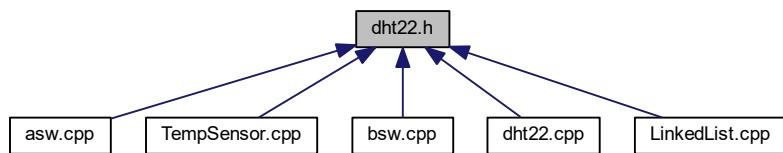
Pointer to `dht22` driver object

Definition at line 20 of file `dht22.cpp`.

## 4.12 dht22.h File Reference

DHT22 driver header file.

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



### Classes

- class `dht22`

*DHT 22 driver class.*

### Variables

- `dht22 * p_global_BSW_dht22`

#### 4.12.1 Detailed Description

DHT22 driver header file.

#### Date

23 mars 2018

#### Author

nicls67

## 4.12.2 Variable Documentation

### 4.12.2.1 p\_global\_BSW\_dht22

`dht22* p_global_BSW_dht22`

Pointer to `dht22` driver object

Definition at line 20 of file `dht22.cpp`.

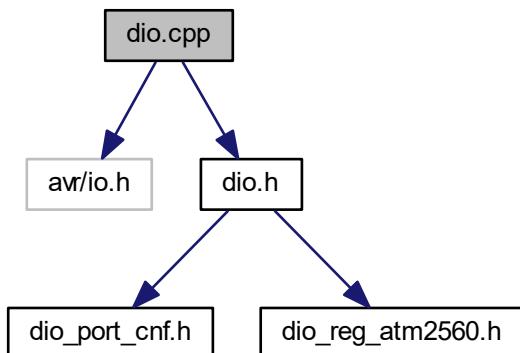
## 4.13 dio.cpp File Reference

DIO library.

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

```
#include "dio.h"
```

Include dependency graph for `dio.cpp`:



## Variables

- `dio * p_global_BSW_dio`

## 4.13.1 Detailed Description

DIO library.

### Date

13 mars 2018

### Author

nicls67

### 4.13.2 Variable Documentation

#### 4.13.2.1 p\_global\_BSW\_dio

`dio* p_global_BSW_dio`

Pointer to dio driver object

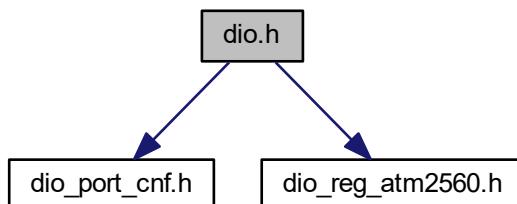
Definition at line 14 of file dio.cpp.

## 4.14 dio.h File Reference

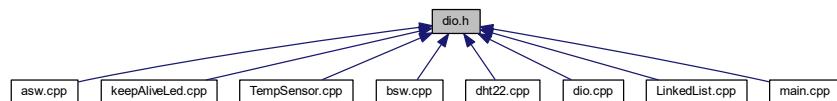
DIO library header file.

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

Include dependency graph for dio.h:



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



## Classes

- class `dio`

*DIO class.*

## Macros

- #define PORT\_CNF\_OUT 1
- #define PORT\_CNF\_IN 0
- #define ENCODE\_PORT(port, pin) (uint8\_t)((((uint8\_t)(port & 0xF)) << 3) | (uint8\_t)(pin & 0x7))
- #define DECODE\_PORT(portcode) (uint8\_t)((portcode >> 3) & 0xF)
- #define DECODE\_PIN(portcode) (uint8\_t)(portcode & 0x7)

## Variables

- dio \* p\_global\_BSW\_dio

### 4.14.1 Detailed Description

DIO library header file.

#### Date

13 mars 2018

#### Author

nicls67

### 4.14.2 Macro Definition Documentation

#### 4.14.2.1 DECODE\_PIN

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

Macro used to extract pin index

Definition at line 20 of file dio.h.

#### 4.14.2.2 DECODE\_PORT

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

Macro used to extract port index

Definition at line 19 of file dio.h.

#### 4.14.2.3 ENCODE\_PORT

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

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

Definition at line 18 of file dio.h.

#### 4.14.2.4 PORT\_CNF\_IN

```
#define PORT_CNF_IN 0
```

Pin is configured as input

Definition at line 16 of file dio.h.

#### 4.14.2.5 PORT\_CNF\_OUT

```
#define PORT_CNF_OUT 1
```

Pin is configured as output

Definition at line 15 of file dio.h.

### 4.14.3 Variable Documentation

#### 4.14.3.1 p\_global\_BSW\_dio

```
dio* p_global_BSW_dio
```

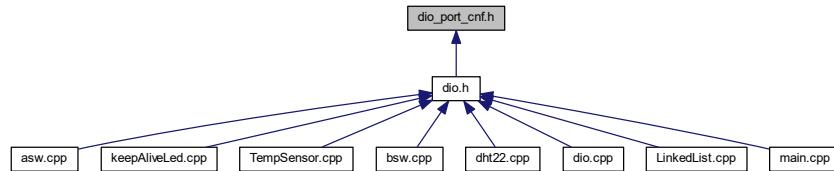
Pointer to dio driver object

Definition at line 14 of file dio.cpp.

## 4.15 dio\_port\_cnf.h File Reference

Digital ports configuration file.

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



### Macros

- `#define PORTB_CNF_DDRB (uint8_t)0b11000000`  
*Defines the configuration of DDRB register.*
- `#define PORTB_CNF_PORTB (uint8_t)0b01010000`  
*Defines the configuration of PORTB register.*
- `#define PORT_A 0`
- `#define PORT_B 1`
- `#define PORT_C 2`
- `#define PORT_D 3`

### 4.15.1 Detailed Description

Digital ports configuration file.

#### Date

19 mars 2019

#### Author

nicls67

### 4.15.2 Macro Definition Documentation

#### 4.15.2.1 PORT\_A

```
#define PORT_A 0
```

PORTA index

Definition at line 42 of file dio\_port\_cnf.h.

#### 4.15.2.2 PORT\_B

```
#define PORT_B 1
```

PORTB index

Definition at line 43 of file dio\_port\_cnf.h.

#### 4.15.2.3 PORT\_C

```
#define PORT_C 2
```

PORTC index

Definition at line 44 of file dio\_port\_cnf.h.

#### 4.15.2.4 PORT\_D

```
#define PORT_D 3
```

PORTD index

Definition at line 45 of file dio\_port\_cnf.h.

#### 4.15.2.5 PORTB\_CNF\_DDRC

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

Defines the configuration of DDRC register.

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

PB0 : N/A

PB1 : N/A

PB2 : N/A

PB3 : N/A

PB4 : IN

PB5 : N/A

PB6 : OUT

PB7 : OUT

Definition at line 25 of file dio\_port\_cnf.h.

#### 4.15.2.6 PORTB\_CNF\_PORTB

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

Defines the configuration of PORTB register.

This constant defines the initial state of IO pins for PORT B. It will configure register PORTB. For outputs pins, it defines the initial level (high or low). For input pins, it defines if the pins is configured as high-Z or pull-up.

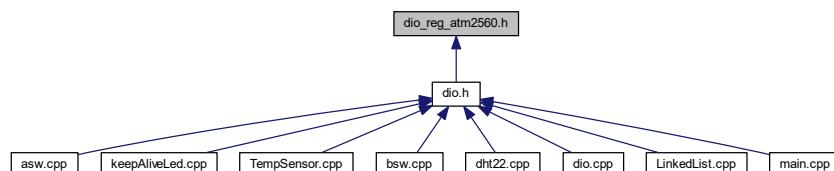
PB0 : N/A  
 PB1 : N/A  
 PB2 : N/A  
 PB3 : N/A  
 PB4 : Pull-up  
 PB5 : N/A  
 PB6 : HIGH  
 PB7 : LOW

Definition at line 40 of file dio\_port\_cnf.h.

## 4.16 dio\_reg\_atm2560.h File Reference

Defines DIO register addresses for ATMEGA2560.

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



## Macros

- #define **PORTA\_PTR** (volatile uint8\_t \*)(0x02 + 0x20)
- #define **PORTB\_PTR** (volatile uint8\_t \*)(0x05 + 0x20)
- #define **PORTC\_PTR** (volatile uint8\_t \*) (0x08 + 0x20)
- #define **PORTD\_PTR** (volatile uint8\_t \*) (0x0B + 0x20)
- #define **PINA\_PTR** (volatile uint8\_t \*) (0x00 + 0x20)
- #define **PINB\_PTR** (volatile uint8\_t \*) (0x03 + 0x20)
- #define **PINC\_PTR** (volatile uint8\_t \*) (0x06 + 0x20)
- #define **PIND\_PTR** (volatile uint8\_t \*) (0x09 + 0x20)
- #define **DDRA\_PTR** (volatile uint8\_t \*) (0x01 + 0x20)
- #define **DDRB\_PTR** (volatile uint8\_t \*) (0x04 + 0x20)
- #define **DDRC\_PTR** (volatile uint8\_t \*) (0x07 + 0x20)
- #define **DDRD\_PTR** (volatile uint8\_t \*) (0x0A + 0x20)

### 4.16.1 Detailed Description

Defines DIO register addresses for ATMEGA2560.

#### Date

19 mars 2019

#### Author

nicls67

### 4.16.2 Macro Definition Documentation

#### 4.16.2.1 DDRA\_PTR

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

Macro defining pointer to DDR A register

Definition at line 24 of file dio\_reg\_atm2560.h.

#### 4.16.2.2 DDRB\_PTR

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

Macro defining pointer to DDR B register

Definition at line 25 of file dio\_reg\_atm2560.h.

#### 4.16.2.3 DDRC\_PTR

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

Macro defining pointer to DDR C register

Definition at line 26 of file dio\_reg\_atm2560.h.

#### 4.16.2.4 DDRD\_PTR

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

Macro defining pointer to DDR D register

Definition at line 27 of file dio\_reg\_atm2560.h.

#### 4.16.2.5 PINA\_PTR

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

Macro defining pointer to PIN A register

Definition at line 19 of file dio\_reg\_atm2560.h.

#### 4.16.2.6 PINB\_PTR

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

Macro defining pointer to PIN B register

Definition at line 20 of file dio\_reg\_atm2560.h.

#### 4.16.2.7 PINC\_PTR

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

Macro defining pointer to PIN C register

Definition at line 21 of file dio\_reg\_atm2560.h.

#### 4.16.2.8 PIND\_PTR

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

Macro defining pointer to PIN D register

Definition at line 22 of file dio\_reg\_atm2560.h.

#### 4.16.2.9 PORTA\_PTR

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

Macro defining pointer to PORT A register

Definition at line 14 of file dio\_reg\_atm2560.h.

#### 4.16.2.10 PORTB\_PTR

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

Macro defining pointer to PORT B register

Definition at line 15 of file dio\_reg\_atm2560.h.

#### 4.16.2.11 PORTC\_PTR

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

Macro defining pointer to PORT C register

Definition at line 16 of file dio\_reg\_atm2560.h.

#### 4.16.2.12 PORTD\_PTR

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

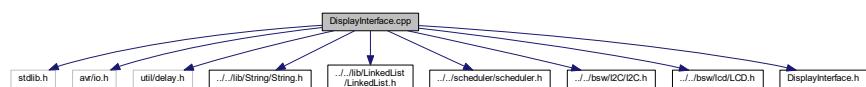
Macro defining pointer to PORT D register

Definition at line 17 of file dio\_reg\_atm2560.h.

## 4.17 DisplayInterface.cpp File Reference

Source code file for display services.

```
#include <stdlib.h>
#include <avr/io.h>
#include <util/delay.h>
#include "../lib/String/String.h"
#include "../lib/LinkedList/LinkedList.h"
#include "../scheduler/scheduler.h"
#include "../bsw/I2C/I2C.h"
#include "../bsw/lcd/LCD.h"
#include "DisplayInterface.h"
Include dependency graph for DisplayInterface.cpp:
```



## Variables

- `DisplayInterface * p_global_ASW_DisplayInterface`

### 4.17.1 Detailed Description

Source code file for display services.

#### Date

23 avr. 2019

#### Author

nicls67

### 4.17.2 Variable Documentation

#### 4.17.2.1 `p_global_ASW_DisplayInterface`

`DisplayInterface* p_global_ASW_DisplayInterface`

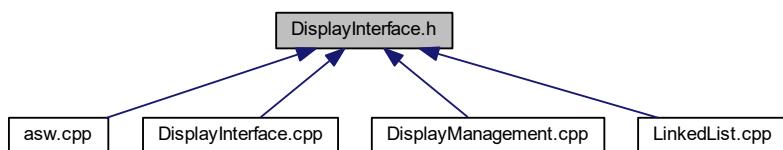
Pointer to `DisplayInterface` object

Definition at line 25 of file `DisplayInterface.cpp`.

## 4.18 DisplayInterface.h File Reference

`DisplayInterface` class header file.

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



## Classes

- struct [T\\_Display\\_shift\\_data](#)  
*Structure containing shift data.*
- struct [T\\_display\\_data](#)  
*Structure containing display data.*
- class [DisplayInterface](#)  
*Display interface services class.*

## Macros

- `#define DISPLAY_LINE_SHIFT_PERIOD_MS 500`
- `#define DISPLAY_LINE_SHIFT_TEMPO_TIME 6`

## Enumerations

- enum [T\\_DisplayInterface\\_LineDisplayMode](#) { [NORMAL](#), [LINE\\_SHIFT](#), [GO\\_TO\\_NEXT\\_LINE](#) }  
*Modes for line display.*
- enum [T\\_DisplayInterface\\_LineAlignment](#) { [LEFT](#), [CENTER](#), [RIGHT](#) }  
*Alignment mode for line display.*

## Variables

- `DisplayInterface * p_global_ASW_DisplayInterface`

### 4.18.1 Detailed Description

[DisplayInterface](#) class header file.

#### Date

23 avr. 2019

#### Author

nicls67

### 4.18.2 Macro Definition Documentation

#### 4.18.2.1 DISPLAY\_LINE\_SHIFT\_PERIOD\_MS

```
#define DISPLAY_LINE_SHIFT_PERIOD_MS 500
```

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

Definition at line 68 of file [DisplayInterface.h](#).

#### 4.18.2.2 DISPLAY\_LINE\_SHIFT\_TEMPO\_TIME

```
#define DISPLAY_LINE_SHIFT_TEMPO_TIME 6
```

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

Definition at line 69 of file DisplayInterface.h.

### 4.18.3 Enumeration Type Documentation

#### 4.18.3.1 T\_DisplayInterface\_LineAlignment

```
enum T_DisplayInterface_LineAlignment
```

Alignment mode for line display.

This enumeration defines the possible alignment mode for the text displayed. It is only used when the display mode is NORMAL or GO\_TO\_NEXT\_LINE.

Enumerator

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

Definition at line 33 of file DisplayInterface.h.

#### 4.18.3.2 T\_DisplayInterface\_LineDisplayMode

```
enum T_DisplayInterface_LineDisplayMode
```

Modes for line display.

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

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

Enumerator

NORMAL	
LINE_SHIFT	
GO_TO_NEXT_LINE	

Definition at line 20 of file DisplayInterface.h.

#### 4.18.4 Variable Documentation

##### 4.18.4.1 p\_global\_ASW\_DisplayInterface

`DisplayInterface* p_global_ASW_DisplayInterface`

Pointer to [DisplayInterface](#) object

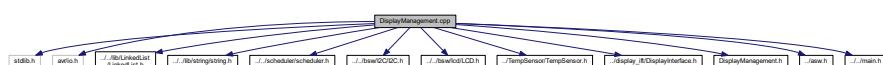
Definition at line 25 of file DisplayInterface.cpp.

## 4.19 DisplayManagement.cpp File Reference

Display management source file.

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

Include dependency graph for DisplayManagement.cpp:



## Variables

- [DisplayManagement \\* p\\_global\\_ASW\\_DisplayManagement](#)

### 4.19.1 Detailed Description

Display management source file.

#### Date

1 mai 2019

#### Author

nicls67

### 4.19.2 Variable Documentation

#### 4.19.2.1 p\_global\_ASW\_DisplayManagement

`DisplayManagement* p_global_ASW_DisplayManagement`

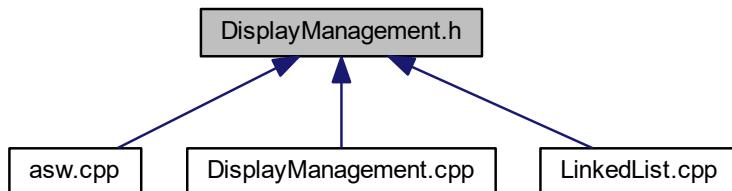
Pointer to [DisplayManagement](#) object

Definition at line 28 of file [DisplayManagement.cpp](#).

## 4.20 DisplayManagement.h File Reference

Display management class header file.

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



### Classes

- class [DisplayManagement](#)

*Display management class.*

### Macros

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

## Variables

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

### 4.20.1 Detailed Description

Display management class header file.

#### Date

1 mai 2019

#### Author

nicls67

### 4.20.2 Macro Definition Documentation

#### 4.20.2.1 DISPLAY\_MGT\_I2C\_BITRATE

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

I2C bus bitrate is 100 kHz

Definition at line 21 of file DisplayManagement.h.

#### 4.20.2.2 DISPLAY\_MGT\_LCD\_I2C\_ADDR

```
#define DISPLAY_MGT_LCD_I2C_ADDR 0x27
```

I2C address of the screen

Definition at line 13 of file DisplayManagement.h.

#### 4.20.2.3 DISPLAY\_MGT\_LINE\_HUM

```
#define DISPLAY_MGT_LINE_HUM 1
```

Current humidity is displayed on line 1

Definition at line 19 of file DisplayManagement.h.

#### 4.20.2.4 DISPLAY\_MGT\_LINE\_TEMP

```
#define DISPLAY_MGT_LINE_TEMP 0
```

Current temperature is displayed on line 0

Definition at line 18 of file DisplayManagement.h.

#### 4.20.2.5 DISPLAY\_MGT\_PERIOD\_TASK\_SENSOR

```
#define DISPLAY_MGT_PERIOD_TASK_SENSOR 1500
```

Display is updated every 1.5s

Definition at line 15 of file DisplayManagement.h.

#### 4.20.2.6 DISPLAY\_MGT\_PERIOD\_WELCOME\_MSG\_REMOVAL

```
#define DISPLAY_MGT_PERIOD_WELCOME_MSG_REMOVAL 5000
```

Time after which one the welcome message is removed

Definition at line 16 of file DisplayManagement.h.

### 4.20.3 Variable Documentation

#### 4.20.3.1 humidityDisplayString

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

[String](#) used for humidity display

Definition at line 43 of file DisplayManagement.h.

#### 4.20.3.2 LCD\_init\_cnf

```
const T\_LCD\_conf\_struct LCD_init_cnf
```

**Initial value:**

```
= {  
    DISPLAY_MGT_I2C_BITRATE,  
    DISPLAY_MGT_LCD_I2C_ADDR,  
    LCD_CNF_BACKLIGHT_ON,  
    LCD_CNF_TWO_LINE,  
    LCD_CNF_FONT_5_8,  
    LCD_CNF_DISPLAY_ON,  
    LCD_CNF_CURSOR_OFF,  
    LCD_CNF_CURSOR_BLINK_OFF,  
    LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT,  
    LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF  
}
```

[LCD](#) configuration structure.

This structure defines the initial configuration of the [LCD](#) screen.

Definition at line 27 of file [DisplayManagement.h](#).

#### 4.20.3.3 noSensorDisplayString

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

[String](#) used in case temperature sensor is deactivated

Definition at line 44 of file [DisplayManagement.h](#).

#### 4.20.3.4 p\_global\_ASW\_DisplayManagement

```
DisplayManagement* p_global_ASW_DisplayManagement
```

Pointer to [DisplayManagement](#) object

Definition at line 28 of file [DisplayManagement.cpp](#).

#### 4.20.3.5 tempDisplayString

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

[String](#) used for temperature display

Definition at line 42 of file [DisplayManagement.h](#).

#### 4.20.3.6 welcomeMessageString

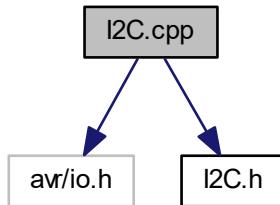
```
const uint8_t welcomeMessageString[] = "Bienvenue !"
```

Definition at line 41 of file DisplayManagement.h.

### 4.21 I2C.cpp File Reference

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

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



#### Variables

- [I2C \\* p\\_global\\_BSW\\_i2c](#)

#### 4.21.1 Detailed Description

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

##### Date

19 avr. 2019

##### Author

nicls67

#### 4.21.2 Variable Documentation

### 4.21.2.1 p\_global\_BSW\_i2c

`I2C* p_global_BSW_i2c`

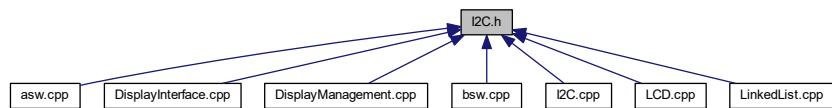
Pointer to `I2C` driver object

Definition at line 14 of file I2C.cpp.

## 4.22 I2C.h File Reference

`I2C` class header file.

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



### Classes

- class `I2C`

*Two-wire serial interface (`I2C`) class definition.*

### Macros

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

### Variables

- `I2C * p_global_BSW_i2c`

### 4.22.1 Detailed Description

`I2C` class header file.

#### Date

19 avr. 2019

#### Author

nicls67

## 4.22.2 Macro Definition Documentation

### 4.22.2.1 DATA\_ACK

```
#define DATA_ACK 0x28
```

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

Definition at line 15 of file I2C.h.

### 4.22.2.2 SLA\_ACK

```
#define SLA_ACK 0x18
```

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

Definition at line 14 of file I2C.h.

### 4.22.2.3 START

```
#define START 0x08
```

TWSR status code : START condition transmitted

Definition at line 13 of file I2C.h.

## 4.22.3 Variable Documentation

### 4.22.3.1 p\_global\_BSW\_i2c

```
I2C* p_global_BSW_i2c
```

Pointer to [I2C](#) driver object

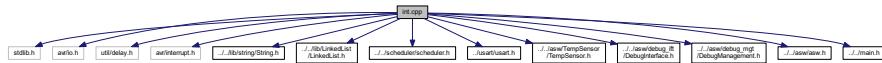
Definition at line 14 of file I2C.cpp.

## 4.23 int.cpp File Reference

Interrupt management source file.

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

Include dependency graph for int.cpp:



## Functions

- **ISR (TIMER1\_COMPA\_vect)**  
*Main software interrupt.*
- **ISR (USART0\_RX\_vect)**  
*USART Rx Complete interrupt.*

### 4.23.1 Detailed Description

Interrupt management source file.

#### Date

22 mai 2019

#### Author

nicls67

### 4.23.2 Function Documentation

#### 4.23.2.1 ISR() [1/2]

```
ISR (
    TIMER1_COMPA_vect )
```

Main software interrupt.

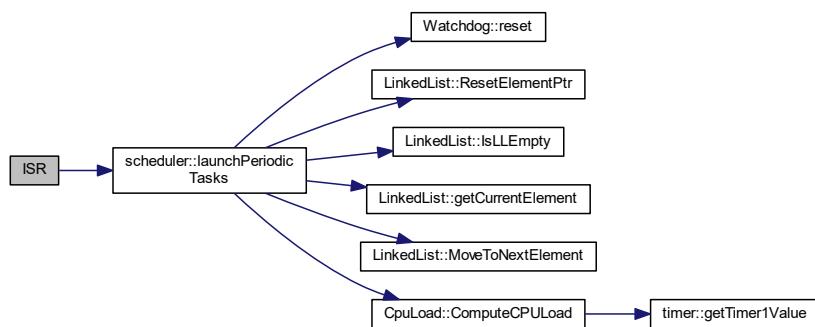
This function handles the interrupt raised by Timer #1. It wakes up the software every 500 ms to perform applications.

##### Returns

Nothing

Definition at line 34 of file int.cpp.

Here is the call graph for this function:



#### 4.23.2.2 ISR() [2/2]

```
ISR (
    USART0_RX_vect )
```

USART Rx Complete interrupt.

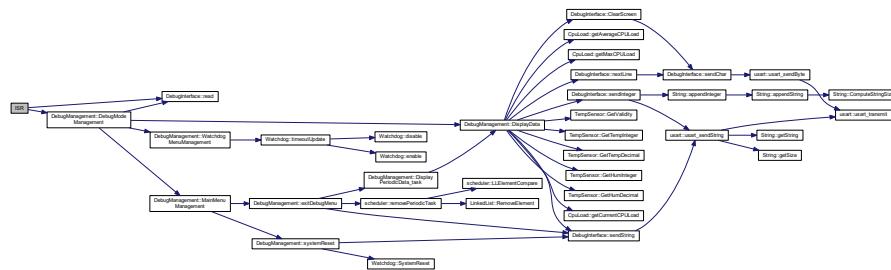
This function handles the interrupt raised when a frame has been received by USART. If debug mode mode is active, it calls debug mode management function. If inactive, it calls debug mode activation function if the received character is 'a'

## Returns

Nothing

Definition at line 46 of file int.cpp.

Here is the call graph for this function:

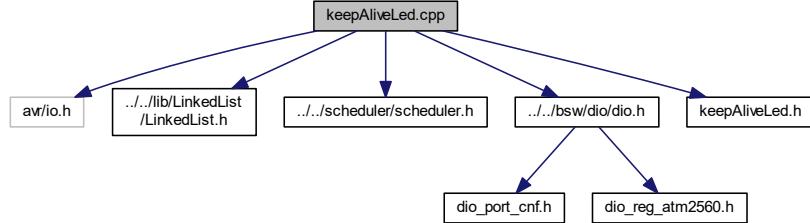


## 4.24 keepAliveLed.cpp File Reference

Definition of function for class `keepAliveLed`.

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

Include dependency graph for keepAliveLed.cpp:



## Variables

- `keepAliveLed * p_global_ASW_keepAliveLed`

#### 4.24.1 Detailed Description

Definition of function for class keepAliveLed.

Date

17 mars 2018

## Author

nicls67

## 4.24.2 Variable Documentation

### 4.24.2.1 p\_global\_ASW\_keepAliveLed

`keepAliveLed* p_global_ASW_keepAliveLed`

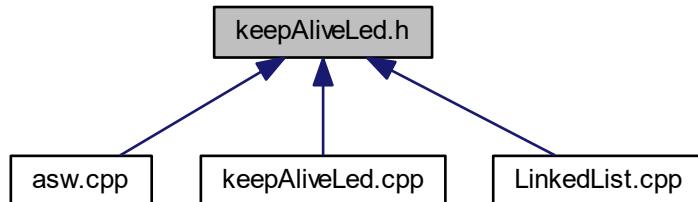
Pointer to [keepAliveLed](#) object

Definition at line 20 of file [keepAliveLed.cpp](#).

## 4.25 keepAliveLed.h File Reference

Class [keepAliveLed](#) header file.

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



### Classes

- class [keepAliveLed](#)  
*Class for keep-alive LED blinking.*

### Macros

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

### Variables

- `keepAliveLed * p_global_ASW_keepAliveLed`

### 4.25.1 Detailed Description

Class [keepAliveLed](#) header file.

Date

17 mars 2018

Author

nicls67

### 4.25.2 Macro Definition Documentation

#### 4.25.2.1 LED\_PORT

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

LED is connected to port PB7

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

#### 4.25.2.2 PERIOD\_MS\_TASK\_LED

```
#define PERIOD_MS_TASK_LED SW_PERIOD_MS
```

Period for led blinking

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

### 4.25.3 Variable Documentation

#### 4.25.3.1 p\_global\_ASW\_keepAliveLed

```
keepAliveLed* p_global_ASW_keepAliveLed
```

Pointer to [keepAliveLed](#) object

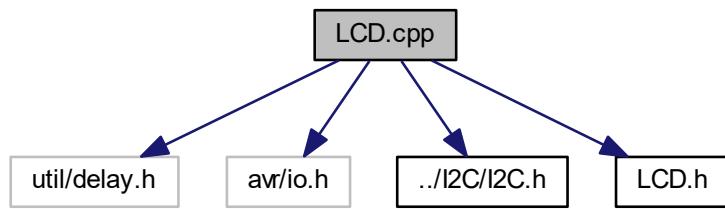
Definition at line 20 of file [keepAliveLed.cpp](#).

## 4.26 LCD.cpp File Reference

[LCD](#) class source file.

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

Include dependency graph for LCD.cpp:
```



### Variables

- [LCD \\* p\\_global\\_BSW\\_lcd](#)

#### 4.26.1 Detailed Description

[LCD](#) class source file.

##### Date

20 avr. 2019

##### Author

nicls67

#### 4.26.2 Variable Documentation

##### 4.26.2.1 p\_global\_BSW\_lcd

`LCD* p_global_BSW_lcd`

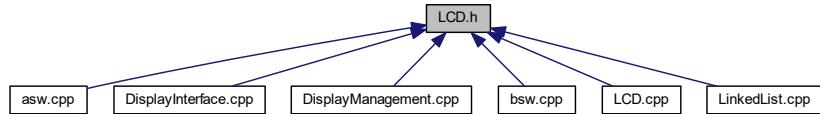
Pointer to [LCD](#) driver object

Definition at line 16 of file `LCD.cpp`.

## 4.27 LCD.h File Reference

LCD class header file.

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



### Classes

- struct [T\\_LCD\\_conf\\_struct](#)  
*Structure defining LCD configuration.*
- class [LCD](#)  
*Class for LCD S2004A display driver.*

### Macros

- #define [EN\\_PIN](#) 2
- #define [RW\\_PIN](#) 1
- #define [RS\\_PIN](#) 0
- #define [BACKLIGHT\\_PIN](#) 3
- #define [LCD\\_INST\\_CLR\\_DISPLAY\\_BIT](#) 0
- #define [LCD\\_INST\\_FUNCTION\\_SET](#) 5
- #define [LCD\\_INST\\_DISPLAY\\_CTRL](#) 3
- #define [LCD\\_INST\\_ENTRY\\_MODE\\_SET](#) 2
- #define [LCD\\_INST\\_SET\\_DDRAM\\_ADDR](#) 7
- #define [LCD\\_FCT\\_SET\\_FIELD\\_DL](#) 4
- #define [LCD\\_FCT\\_SET\\_FIELD\\_N](#) 3
- #define [LCD\\_FCT\\_SET\\_FIELD\\_F](#) 2
- #define [LCD\\_DISPLAY\\_CTRL\\_FIELD\\_D](#) 2
- #define [LCD\\_DISPLAY\\_CTRL\\_FIELD\\_C](#) 1
- #define [LCD\\_DISPLAY\\_CTRL\\_FIELD\\_B](#) 0
- #define [LCD\\_CNF\\_SHIFT\\_ID](#) 1
- #define [LCD\\_CNF\\_SHIFT\\_SH](#) 0
- #define [LCD\\_CNF\\_ONE\\_LINE](#) 0
- #define [LCD\\_CNF\\_TWO\\_LINE](#) 1
- #define [LCD\\_CNF\\_FONT\\_5\\_8](#) 0
- #define [LCD\\_CNF\\_FONT\\_5\\_11](#) 1
- #define [LCD\\_CNF\\_DISPLAY\\_ON](#) 1
- #define [LCD\\_CNF\\_DISPLAY\\_OFF](#) 0
- #define [LCD\\_CNF\\_CURSOR\\_ON](#) 1
- #define [LCD\\_CNF\\_CURSOR\\_OFF](#) 0
- #define [LCD\\_CNF\\_CURSOR\\_BLINK\\_ON](#) 1
- #define [LCD\\_CNF\\_CURSOR\\_BLINK\\_OFF](#) 0
- #define [LCD\\_CNF\\_ENTRY\\_MODE\\_DIRECTION\\_RIGHT](#) 1

- `#define LCD_CNF_ENTRY_MODE_DIRECTION_LEFT 0`
- `#define LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON 1`
- `#define LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF 0`
- `#define LCD_CNF_BACKLIGHT_ON 1`
- `#define LCD_CNF_BACKLIGHT_OFF 0`
- `#define LCD_RAM_1_LINE_MIN 0`
- `#define LCD_RAM_1_LINE_MAX 0x4F`
- `#define LCD_RAM_2_LINES_MIN_1 0`
- `#define LCD_RAM_2_LINES_MAX_1 0x27`
- `#define LCD_RAM_2_LINES_MIN_2 0x40`
- `#define LCD_RAM_2_LINES_MAX_2 0x67`
- `#define LCD_WAIT_CLR_RETURN 1600`
- `#define LCD_WAIT_OTHER_MODES 40`
- `#define LCD_SIZE_NB_CHAR_PER_LINE 20`
- `#define LCD_SIZE_NB_LINES 4`

## Enumerations

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

## Variables

- `LCD * p_global_BSW_lcd`

### 4.27.1 Detailed Description

`LCD` class header file.

#### Date

20 avr. 2019

#### Author

nicls67

### 4.27.2 Macro Definition Documentation

#### 4.27.2.1 BACKLIGHT\_PIN

```
#define BACKLIGHT_PIN 3
```

Backlight pin is on P3

Definition at line 17 of file LCD.h.

#### 4.27.2.2 EN\_PIN

```
#define EN_PIN 2
```

EN bit is on P2

Definition at line 14 of file LCD.h.

#### 4.27.2.3 LCD\_CNF\_BACKLIGHT\_OFF

```
#define LCD_CNF_BACKLIGHT_OFF 0
```

Backlight is disabled

Definition at line 70 of file LCD.h.

#### 4.27.2.4 LCD\_CNF\_BACKLIGHT\_ON

```
#define LCD_CNF_BACKLIGHT_ON 1
```

Backlight is enabled

Definition at line 69 of file LCD.h.

#### 4.27.2.5 LCD\_CNF\_CURSOR\_BLINK\_OFF

```
#define LCD_CNF_CURSOR_BLINK_OFF 0
```

Cursor blinking is off, bit is set to 0

Definition at line 58 of file LCD.h.

#### 4.27.2.6 LCD\_CNF\_CURSOR\_BLINK\_ON

```
#define LCD_CNF_CURSOR_BLINK_ON 1
```

Cursor blinking is on, bit is set to 1

Definition at line 57 of file LCD.h.

#### 4.27.2.7 LCD\_CNF\_CURSOR\_OFF

```
#define LCD_CNF_CURSOR_OFF 0
```

Cursor is off, bit is set to 0

Definition at line 54 of file LCD.h.

#### 4.27.2.8 LCD\_CNF\_CURSOR\_ON

```
#define LCD_CNF_CURSOR_ON 1
```

Cursor is on, bit is set to 1

Definition at line 53 of file LCD.h.

#### 4.27.2.9 LCD\_CNF\_DISPLAY\_OFF

```
#define LCD_CNF_DISPLAY_OFF 0
```

Display is off, bit is set to 0

Definition at line 50 of file LCD.h.

#### 4.27.2.10 LCD\_CNF\_DISPLAY\_ON

```
#define LCD_CNF_DISPLAY_ON 1
```

Display is on, bit is set to 1

Definition at line 49 of file LCD.h.

**4.27.2.11 LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_LEFT**

```
#define LCD_CNF_ENTRY_MODE_DIRECTION_LEFT 0
```

Direction of shift is left, bit is set to 0

Definition at line 62 of file LCD.h.

**4.27.2.12 LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_RIGHT**

```
#define LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT 1
```

Direction of shift is right, bit is set to 1

Definition at line 61 of file LCD.h.

**4.27.2.13 LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT\_OFF**

```
#define LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF 0
```

Display shift is not performed, bit is set to 0

Definition at line 66 of file LCD.h.

**4.27.2.14 LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT\_ON**

```
#define LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON 1
```

Display shift is performed, bit is set to 1

Definition at line 65 of file LCD.h.

**4.27.2.15 LCD\_CNF\_FONT\_5\_11**

```
#define LCD_CNF_FONT_5_11 1
```

Two-line configuration, bit is set to 1

Definition at line 46 of file LCD.h.

#### 4.27.2.16 LCD\_CNF\_FONT\_5\_8

```
#define LCD_CNF_FONT_5_8 0
```

One-line configuration, bit is set to 0

Definition at line 45 of file LCD.h.

#### 4.27.2.17 LCD\_CNF\_ONE\_LINE

```
#define LCD_CNF_ONE_LINE 0
```

One-line configuration, bit is set to 0

Definition at line 41 of file LCD.h.

#### 4.27.2.18 LCD\_CNF\_SHIFT\_ID

```
#define LCD_CNF_SHIFT_ID 1
```

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

Definition at line 37 of file LCD.h.

#### 4.27.2.19 LCD\_CNF\_SHIFT\_SH

```
#define LCD_CNF_SHIFT_SH 0
```

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

Definition at line 38 of file LCD.h.

#### 4.27.2.20 LCD\_CNF\_TWO\_LINE

```
#define LCD_CNF_TWO_LINE 1
```

Two-line configuration, bit is set to 1

Definition at line 42 of file LCD.h.

**4.27.2.21 LCD\_DISPLAY\_CTRL\_FIELD\_B**

```
#define LCD_DISPLAY_CTRL_FIELD_B 0
```

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

Definition at line 34 of file LCD.h.

**4.27.2.22 LCD\_DISPLAY\_CTRL\_FIELD\_C**

```
#define LCD_DISPLAY_CTRL_FIELD_C 1
```

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

Definition at line 33 of file LCD.h.

**4.27.2.23 LCD\_DISPLAY\_CTRL\_FIELD\_D**

```
#define LCD_DISPLAY_CTRL_FIELD_D 2
```

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

Definition at line 32 of file LCD.h.

**4.27.2.24 LCD\_FCT\_SET\_FIELD\_DL**

```
#define LCD_FCT_SET_FIELD_DL 4
```

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

Definition at line 27 of file LCD.h.

**4.27.2.25 LCD\_FCT\_SET\_FIELD\_F**

```
#define LCD_FCT_SET_FIELD_F 2
```

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

Definition at line 29 of file LCD.h.

#### 4.27.2.26 LCD\_FCT\_SET\_FIELD\_N

```
#define LCD_FCT_SET_FIELD_N 3
```

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

Definition at line 28 of file LCD.h.

#### 4.27.2.27 LCD\_INST\_CLR\_DISPLAY\_BIT

```
#define LCD_INST_CLR_DISPLAY_BIT 0
```

Instruction bit for "clear display" is DB0

Definition at line 20 of file LCD.h.

#### 4.27.2.28 LCD\_INST\_DISPLAY\_CTRL

```
#define LCD_INST_DISPLAY_CTRL 3
```

Instruction bit for "display control" is DB3

Definition at line 22 of file LCD.h.

#### 4.27.2.29 LCD\_INST\_ENTRY\_MODE\_SET

```
#define LCD_INST_ENTRY_MODE_SET 2
```

Instruction bit for "entry mode" is DB2

Definition at line 23 of file LCD.h.

#### 4.27.2.30 LCD\_INST\_FUNCTION\_SET

```
#define LCD_INST_FUNCTION_SET 5
```

Instruction bit for "function set" is DB5

Definition at line 21 of file LCD.h.

**4.27.2.31 LCD\_INST\_SET\_DDRAM\_ADDR**

```
#define LCD_INST_SET_DDRAM_ADDR 7
```

Instruction bit for "set DDRAM address" is DB7

Definition at line 24 of file LCD.h.

**4.27.2.32 LCD\_RAM\_1\_LINE\_MAX**

```
#define LCD_RAM_1_LINE_MAX 0x4F
```

Maximum address value in 1-line mode

Definition at line 74 of file LCD.h.

**4.27.2.33 LCD\_RAM\_1\_LINE\_MIN**

```
#define LCD_RAM_1_LINE_MIN 0
```

Minimum address value in 1-line mode

Definition at line 73 of file LCD.h.

**4.27.2.34 LCD\_RAM\_2\_LINES\_MAX\_1**

```
#define LCD_RAM_2_LINES_MAX_1 0x27
```

Maximum address value in 2-lines mode for line 1

Definition at line 76 of file LCD.h.

**4.27.2.35 LCD\_RAM\_2\_LINES\_MAX\_2**

```
#define LCD_RAM_2_LINES_MAX_2 0x67
```

Maximum address value in 2-lines mode for line 2

Definition at line 78 of file LCD.h.

#### 4.27.2.36 LCD\_RAM\_2\_LINES\_MIN\_1

```
#define LCD_RAM_2_LINES_MIN_1 0
```

Minimum address value in 2-lines mode for line 1

Definition at line 75 of file LCD.h.

#### 4.27.2.37 LCD\_RAM\_2\_LINES\_MIN\_2

```
#define LCD_RAM_2_LINES_MIN_2 0x40
```

Minimum address value in 2-lines mode for line 2

Definition at line 77 of file LCD.h.

#### 4.27.2.38 LCD\_SIZE\_NB\_CHAR\_PER\_LINE

```
#define LCD_SIZE_NB_CHAR_PER_LINE 20
```

LCD screen has 20 characters per line

Definition at line 85 of file LCD.h.

#### 4.27.2.39 LCD\_SIZE\_NB\_LINES

```
#define LCD_SIZE_NB_LINES 4
```

LCD screen has 4 lines

Definition at line 86 of file LCD.h.

#### 4.27.2.40 LCD\_WAIT\_CLR\_RETURN

```
#define LCD_WAIT_CLR_RETURN 1600
```

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

Definition at line 81 of file LCD.h.

#### 4.27.2.41 LCD\_WAIT\_OTHER\_MODES

```
#define LCD_WAIT_OTHER_MODES 40
```

Waiting time after all other modes is at least 38 us

Definition at line 82 of file LCD.h.

#### 4.27.2.42 RS\_PIN

```
#define RS_PIN 0
```

RS pin is on P0

Definition at line 16 of file LCD.h.

#### 4.27.2.43 RW\_PIN

```
#define RW_PIN 1
```

RW pin is on P1

Definition at line 15 of file LCD.h.

### 4.27.3 Enumeration Type Documentation

#### 4.27.3.1 T\_LCD\_command

```
enum T_LCD_command
```

LCD commands enumeration.

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

Enumerator

LCD_CMD_FUNCTION_SET	
LCD_CMD_CLEAR_DISPLAY	
LCD_CMD_DISPLAY_CTRL	
LCD_CMD_ENTRY_MODE_SET	
LCD_CMD_SET_DDRAM_ADDR	

Definition at line 93 of file LCD.h.

#### 4.27.3.2 T\_LCD\_config\_mode

`enum T_LCD_config_mode`

LCD modes enumeration.

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

Enumerator

LCD_MODE_INSTRUCTION	
LCD_MODE_DATA	

Definition at line 107 of file LCD.h.

#### 4.27.3.3 T\_LCD\_ram\_area

`enum T_LCD_ram_area`

Screen RAM definition.

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

Enumerator

LCD_DATA_DDRAM	
LCD_DATA_CGRAM	

Definition at line 118 of file LCD.h.

### 4.27.4 Variable Documentation

#### 4.27.4.1 p\_global\_BSW\_lcd

`LCD* p_global_BSW_lcd`

Pointer to [LCD](#) driver object

Definition at line 16 of file LCD.cpp.

## 4.28 LinkedList.cpp File Reference

Linked List library source file.

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

Include dependency graph for LinkedList.cpp:



### 4.28.1 Detailed Description

Linked List library source file.

#### Date

27 avr. 2019

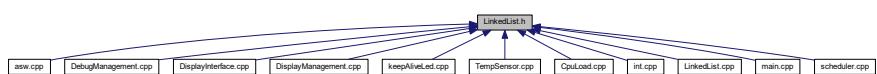
#### Author

nicls67

## 4.29 LinkedList.h File Reference

Linked List library header file.

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



## Classes

- class [LinkedList](#)  
*Linked list class.*
- struct [LinkedList::T\\_LL\\_element](#)  
*Type defining a linked list element.*

## TypeDefs

- typedef [bool\(\\* CompareFctPtr\\_t\)](#) (`void *LLElement, void *CompareElement`)

### 4.29.1 Detailed Description

Linked List library header file.

#### Date

27 avr. 2019

#### Author

nicls67

### 4.29.2 Typedef Documentation

#### 4.29.2.1 CompareFctPtr\_t

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

Type defining a pointer to the comparison function

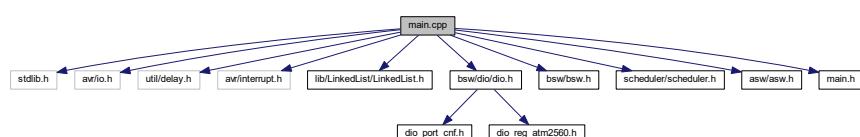
Definition at line 14 of file `LinkedList.h`.

## 4.30 main.cpp File Reference

Background task file.

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

Include dependency graph for `main.cpp`:



## Macros

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

## Functions

- `int main (void)`  
*Background task of program.*

## Variables

- `bool isDebugModeActivated`
- `const T_ASW_init_cnf ASW_init_cnf`

### 4.30.1 Detailed Description

Background task file.

#### Date

12 mars 2018

#### Author

nicls67

### 4.30.2 Macro Definition Documentation

#### 4.30.2.1 DEBUG\_ACTIVE\_PORT

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

Debug activation pin is port PB6

Definition at line 26 of file main.cpp.

### 4.30.3 Function Documentation

#### 4.30.3.1 main()

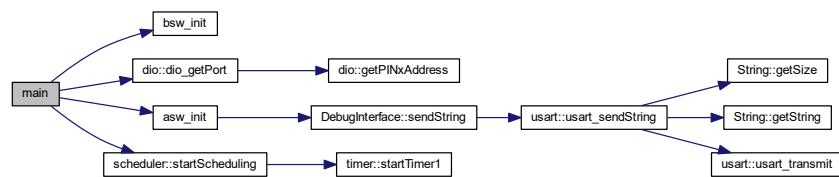
```
int main (
    void )
```

Background task of program.

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

Definition at line 45 of file main.cpp.

Here is the call graph for this function:



#### 4.30.4 Variable Documentation

##### 4.30.4.1 ASW\_init\_cnf

```
const T_ASW_init_cnf ASW_init_cnf
```

**Initial value:**

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

Definition of needed ASW services

Definition at line 30 of file main.cpp.

##### 4.30.4.2 isDebugEnabled

```
bool isDebugEnabled
```

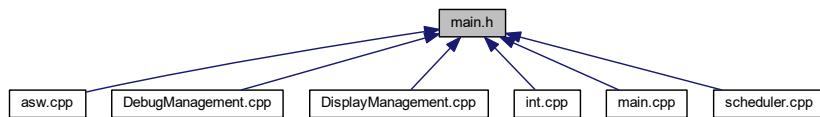
Flag indicating if the debug mode is activated or not

Definition at line 28 of file main.cpp.

## 4.31 main.h File Reference

Background task header file.

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



### Variables

- bool `isDebugModeActivated`
- const `T_ASW_init_cnf ASW_init_cnf`

#### 4.31.1 Detailed Description

Background task header file.

##### Date

17 mars 2018

##### Author

nicls67

#### 4.31.2 Variable Documentation

##### 4.31.2.1 ASW\_init\_cnf

```
const T_ASW_init_cnf ASW_init_cnf
```

Definition of needed ASW services

Definition at line 30 of file main.cpp.

#### 4.31.2.2 isDebugEnabledActivated

```
bool isDebugEnabledActivated
```

Flag indicating if the debug mode is activated or not

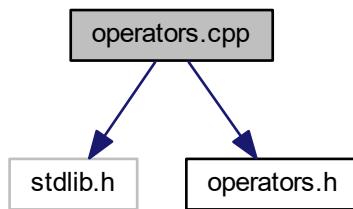
Definition at line 28 of file main.cpp.

## 4.32 operators.cpp File Reference

### c++ operators definitions

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

Include dependency graph for operators.cpp:



### Functions

- void \* [operator new](#) (size\_t a\_size)  
*Operator new.*
- void [operator delete](#) (void \*ptr)  
*Operator delete.*

### 4.32.1 Detailed Description

#### c++ operators definitions

#### Date

14 mars 2018

#### Author

nicls67

## 4.32.2 Function Documentation

### 4.32.2.1 operator delete()

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

Operator delete.

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

#### Parameters

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

#### Returns

Nothing

Definition at line 18 of file operators.cpp.

### 4.32.2.2 operator new()

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

Operator new.

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

#### Parameters

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

#### Returns

Pointer to the start of allocated memory zone

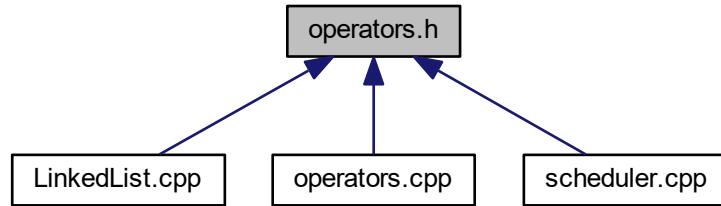
Definition at line 13 of file operators.cpp.

## 4.33 operators.h File Reference

C++ operators definitions header file

---

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



## Functions

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

### 4.33.1 Detailed Description

c++ operators definitions header file

#### Date

14 mars 2018

#### Author

nicls67

### 4.33.2 Function Documentation

#### 4.33.2.1 operator delete()

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

Operator delete.

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

**Parameters**

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

**Returns**

Nothing

Definition at line 18 of file operators.cpp.

**4.33.2.2 operator new()**

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

Operator new.

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

**Parameters**

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

**Returns**

Pointer to the start of allocated memory zone

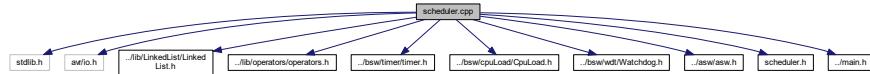
Definition at line 13 of file operators.cpp.

**4.34 scheduler.cpp File Reference**

Defines scheduler class.

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

Include dependency graph for scheduler.cpp:



## Variables

- `scheduler * p_global_scheduler`

### 4.34.1 Detailed Description

Defines scheduler class.

#### Date

16 mars 2018

#### Author

nicls67

### 4.34.2 Variable Documentation

#### 4.34.2.1 `p_global_scheduler`

`scheduler* p_global_scheduler`

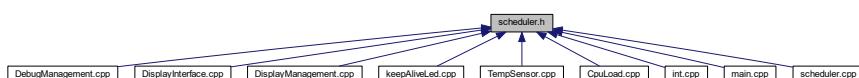
Pointer to scheduler object

Definition at line 27 of file scheduler.cpp.

## 4.35 scheduler.h File Reference

Scheduler class header file.

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



## Classes

- class `scheduler`  
*Scheduler class.*
- struct `scheduler::Task_t`  
*Type defining a task structure.*

## Macros

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

## Typedefs

- `typedef void(* TaskPtr_t) (void)`  
*Type defining a pointer to function.*

## Variables

- `scheduler * p_global_scheduler`

### 4.35.1 Detailed Description

Scheduler class header file.

#### Date

16 mars 2018

#### Author

nicls67

### 4.35.2 Macro Definition Documentation

#### 4.35.2.1 PRESCALER\_PERIODIC\_TIMER

```
#define PRESCALER_PERIODIC_TIMER 256
```

Value of prescaler to use for periodic timer

Definition at line 16 of file scheduler.h.

#### 4.35.2.2 SW\_PERIOD\_MS

```
#define SW_PERIOD_MS 500
```

Software period, used to define periodic timer interrupt

Definition at line 15 of file scheduler.h.

### 4.35.2.3 TIMER\_CTC\_VALUE

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

Compare value for periodic timer

Definition at line 17 of file scheduler.h.

### 4.35.3 Typedef Documentation

#### 4.35.3.1 TaskPtr\_t

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

Type defining a pointer to function.

Definition at line 22 of file scheduler.h.

### 4.35.4 Variable Documentation

#### 4.35.4.1 p\_global\_scheduler

```
scheduler* p_global_scheduler
```

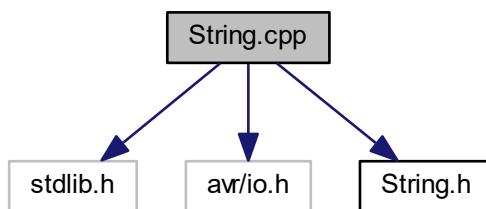
Pointer to scheduler object

Definition at line 27 of file scheduler.cpp.

## 4.36 String.cpp File Reference

[String](#) class source file.

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



### 4.36.1 Detailed Description

[String](#) class source file.

Date

2 mai 2019

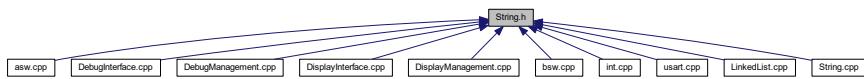
Author

nicls67

## 4.37 String.h File Reference

[String](#) class header file.

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



### Classes

- class [String](#)

*String management class.*

### 4.37.1 Detailed Description

[String](#) class header file.

Date

2 mai 2019

Author

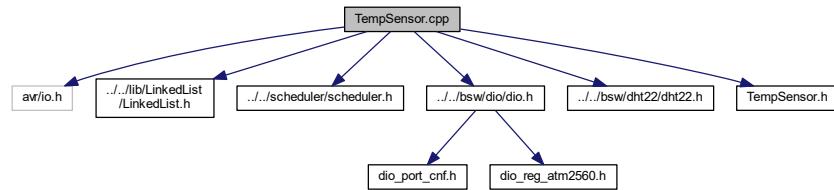
nicls67

## 4.38 TempSensor.cpp File Reference

Defines function of class [TempSensor](#).

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

Include dependency graph for TempSensor.cpp:



### Macros

- [#define PIT\\_BEFORE\\_INVALID 60](#)

### Variables

- [TempSensor \\* p\\_global\\_ASW\\_TempSensor](#)

#### 4.38.1 Detailed Description

Defines function of class [TempSensor](#).

##### Date

23 mars 2018

##### Author

nicls67

#### 4.38.2 Macro Definition Documentation

#### 4.38.2.1 PIT\_BEFORE\_INVALID

```
#define PIT_BEFORE_INVALID 60
```

Definition at line 20 of file TempSensor.cpp.

#### 4.38.3 Variable Documentation

##### 4.38.3.1 p\_global\_ASW\_TempSensor

```
TempSensor* p_global_ASW_TempSensor
```

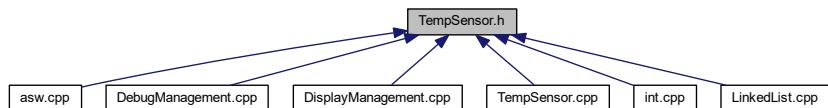
Pointer to [TempSensor](#) object

Definition at line 22 of file TempSensor.cpp.

## 4.39 TempSensor.h File Reference

Class [TempSensor](#) header file.

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



### Classes

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

### Macros

- `#define PERIOD_MS_TASK_TEMP_SENSOR 5000`
- `#define DHT22_PORT_ENCODE_PORT(PORT_B, 6)`

### Variables

- [TempSensor \\* p\\_global\\_ASW\\_TempSensor](#)

#### 4.39.1 Detailed Description

Class [TempSensor](#) header file.

Date

23 mars 2018

Author

nicls67

#### 4.39.2 Macro Definition Documentation

##### 4.39.2.1 DHT22\_PORT

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

DHT22 is connected to port PB6

Definition at line 15 of file TempSensor.h.

##### 4.39.2.2 PERIOD\_MS\_TASK\_TEMP\_SENSOR

```
#define PERIOD_MS_TASK_TEMP_SENSOR 5000
```

Period for reading temperature data

Definition at line 13 of file TempSensor.h.

#### 4.39.3 Variable Documentation

##### 4.39.3.1 p\_global\_ASW\_TempSensor

```
TempSensor* p_global_ASW_TempSensor
```

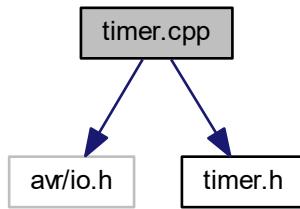
Pointer to [TempSensor](#) object

Definition at line 22 of file TempSensor.cpp.

## 4.40 timer.cpp File Reference

Defines function for class timer.

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



### Variables

- `timer * p_global_BSW_timer`

#### 4.40.1 Detailed Description

Defines function for class timer.

##### Date

15 mars 2018

##### Author

nicls67

#### 4.40.2 Variable Documentation

##### 4.40.2.1 `p_global_BSW_timer`

`timer* p_global_BSW_timer`

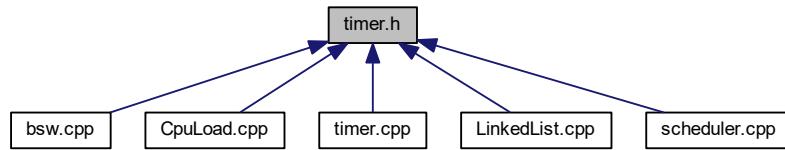
Pointer to timer driver object

Definition at line 13 of file timer.cpp.

## 4.41 timer.h File Reference

Timer class header file.

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



### Classes

- class [timer](#)  
*Class defining a timer.*

### Variables

- [timer \\* p\\_global\\_BSW\\_timer](#)

#### 4.41.1 Detailed Description

Timer class header file.

##### Date

15 mars 2018

##### Author

nicls67

#### 4.41.2 Variable Documentation

##### 4.41.2.1 [p\\_global\\_BSW\\_timer](#)

[timer\\*](#) [p\\_global\\_BSW\\_timer](#)

Pointer to timer driver object

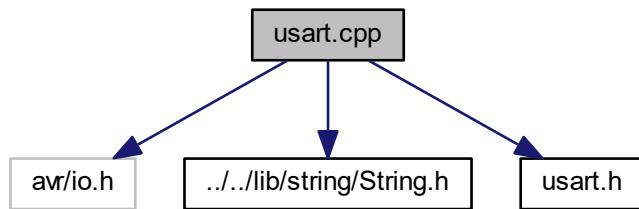
Definition at line 13 of file [timer.cpp](#).

## 4.42 usart.cpp File Reference

BSW library for USART.

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

Include dependency graph for usart.cpp:



### Variables

- `usart * p_global_BSW_usart`

#### 4.42.1 Detailed Description

BSW library for USART.

##### Date

13 mars 2018

##### Author

nicls67

#### 4.42.2 Variable Documentation

##### 4.42.2.1 `p_global_BSW_usart`

`usart* p_global_BSW_usart`

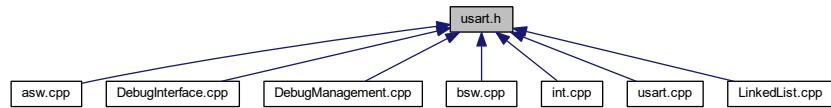
Pointer to usart driver object

Definition at line 16 of file usart.cpp.

## 4.43 usart.h File Reference

Header file for USART library.

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



### Classes

- class [usart](#)  
*USART serial bus class.*

### Variables

- [usart \\* p\\_global\\_BSW\\_usart](#)

#### 4.43.1 Detailed Description

Header file for USART library.

##### Date

13 mars 2018

##### Author

nicls67

#### 4.43.2 Variable Documentation

##### 4.43.2.1 [p\\_global\\_BSW\\_usart](#)

`usart* p_global_BSW_usart`

Pointer to usart driver object

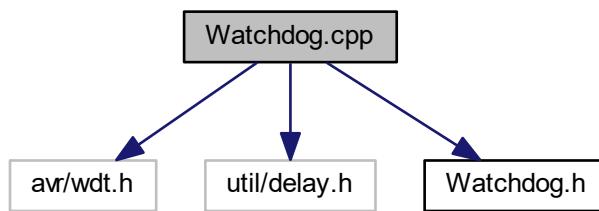
Definition at line 16 of file `usart.cpp`.

## 4.44 Watchdog.cpp File Reference

Class [Watchdog](#) source code file.

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

Include dependency graph for Watchdog.cpp:



### Macros

- `#define WDG_TIMEOUT_DEFAULT_MS WDG_TMO_500MS`

### Variables

- `Watchdog * p_global_BSW_wdg`

#### 4.44.1 Detailed Description

Class [Watchdog](#) source code file.

##### Date

6 juin 2019

##### Author

nicls67

#### 4.44.2 Macro Definition Documentation

#### 4.44.2.1 WDG\_TIMEOUT\_DEFAULT\_MS

```
#define WDG_TIMEOUT_DEFAULT_MS WDG_TMO_500MS
```

Default timeout value is set to 500 ms

Definition at line 18 of file Watchdog.cpp.

### 4.44.3 Variable Documentation

#### 4.44.3.1 p\_global\_BSW\_wdg

```
Watchdog* p_global_BSW_wdg
```

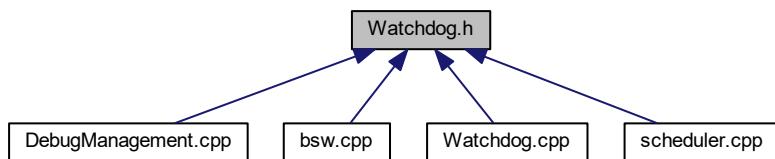
Pointer to [Watchdog](#) driver object

Definition at line 20 of file Watchdog.cpp.

## 4.45 Watchdog.h File Reference

Class [Watchdog](#) header file.

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



### Classes

- class [Watchdog](#)

*Watchdog management class.*

## Macros

- `#define WDG_TMO_15MS WDTO_15MS`  
*Definition of available timeout values.*
- `#define WDG_TMO_30MS WDTO_30MS`
- `#define WDG_TMO_60MS WDTO_60MS`
- `#define WDG_TMO_120MS WDTO_120MS`
- `#define WDG_TMO_250MS WDTO_250MS`
- `#define WDG_TMO_500MS WDTO_500MS`
- `#define WDG_TMO_1S WDTO_1S`
- `#define WDG_TMO_2S WDTO_2S`
- `#define WDG_TMO_4S WDTO_4S`
- `#define WDG_TMO_8S WDTO_8S`

## Variables

- `Watchdog * p_global_BSW_wdg`

### 4.45.1 Detailed Description

Class `Watchdog` header file.

#### Date

6 juin 2019

#### Author

nicls67

### 4.45.2 Macro Definition Documentation

#### 4.45.2.1 WDG\_TMO\_120MS

```
#define WDG_TMO_120MS WDTO_120MS
```

Timeout value is 120 ms

Definition at line 19 of file Watchdog.h.

#### 4.45.2.2 WDG\_TMO\_15MS

```
#define WDG_TMO_15MS WDTO_15MS
```

Definition of available timeout values.

Timeout value is 15 ms

Definition at line 16 of file Watchdog.h.

#### 4.45.2.3 WDG\_TMO\_1S

```
#define WDG_TMO_1S WDTO_1S
```

Timeout value is 1 s

Definition at line 22 of file Watchdog.h.

#### 4.45.2.4 WDG\_TMO\_250MS

```
#define WDG_TMO_250MS WDTO_250MS
```

Timeout value is 250 ms

Definition at line 20 of file Watchdog.h.

#### 4.45.2.5 WDG\_TMO\_2S

```
#define WDG_TMO_2S WDTO_2S
```

Timeout value is 2 s

Definition at line 23 of file Watchdog.h.

#### 4.45.2.6 WDG\_TMO\_30MS

```
#define WDG_TMO_30MS WDTO_30MS
```

Timeout value is 30 ms

Definition at line 17 of file Watchdog.h.

#### 4.45.2.7 WDG\_TMO\_4S

```
#define WDG_TMO_4S WDTO_4S
```

Timeout value is 4 s

Definition at line 24 of file Watchdog.h.

#### 4.45.2.8 WDG\_TMO\_500MS

```
#define WDG_TMO_500MS WDTO_500MS
```

Timeout value is 500 ms

Definition at line 21 of file Watchdog.h.

#### 4.45.2.9 WDG\_TMO\_60MS

```
#define WDG_TMO_60MS WDTO_60MS
```

Timeout value is 60 ms

Definition at line 18 of file Watchdog.h.

#### 4.45.2.10 WDG\_TMO\_8S

```
#define WDG_TMO_8S WDTO_8S
```

Timeout value is 8 s

Definition at line 25 of file Watchdog.h.

### 4.45.3 Variable Documentation

#### 4.45.3.1 p\_global\_BSW\_wdg

`Watchdog*` p\_global\_BSW\_wdg

Pointer to `Watchdog` driver object

Definition at line 20 of file Watchdog.cpp.



# Index

~LinkedList  
    LinkedList, 84

~String  
    String, 99

ASW\_init\_cnf  
    main.cpp, 202  
    main.h, 203

addPeriodicTask  
    scheduler, 91

alignment  
    T\_display\_data, 108

appendBool  
    String, 100

appendChar  
    String, 101

appendInteger  
    String, 102

appendString  
    String, 102

asw.cpp, 141  
    asw\_init, 142

asw.h, 143  
    asw\_init, 143

asw\_init  
    asw.cpp, 142  
    asw.h, 143

AttachNewElement  
    LinkedList, 85

avg\_load  
    CpuLoad, 8

BACKLIGHT\_PIN  
    LCD.h, 188

backlight\_en  
    T\_LCD\_conf\_struct, 112

backlight\_enable  
    LCD, 79

BaudRate  
    uart, 135

bitrate  
    I2C, 63

blinkLed\_task  
    keepAliveLed, 64

bsw.cpp, 144  
    bsw\_init, 145

bsw.h, 146  
    bsw\_init, 146

bsw\_init  
    bsw.cpp, 145

bsw.h, 146  
    bsw\_init, 146

bsw\_init  
    bsw.h, 146

bsw.h, 146

Clear  
    String, 103

ClearFullScreen  
    DisplayInterface, 44

ClearLine  
    DisplayInterface, 45

ClearScreen  
    DebugInterface, 12

ClearStringInDataStruct  
    DisplayInterface, 46

cnfCursorBlink  
    LCD, 80

cnfCursorOnOff  
    LCD, 80

cnfDisplayOnOff  
    LCD, 80

cnfEntryModeDir  
    LCD, 80

cnfEntryModeShift  
    LCD, 80

cnfFontType  
    LCD, 81

cnfI2C\_addr  
    LCD, 81

cnfLineNumber  
    LCD, 81

command  
    LCD, 68

CompareFctPtr\_t  
    LinkedList.h, 200

ComputeCPUload  
    CpuLoad, 6

ComputeStringSize  
    String, 104

ConfigureBacklight  
    LCD, 69

ConfigureCursorBlink  
    LCD, 69

ConfigureCursorOnOff  
    LCD, 70

ConfigureDisplayOnOff  
    LCD, 71

ConfigureEntryModeDir  
    LCD, 71

ConfigureEntryModeShift  
    LCD, 72

ConfigureFontType  
    LCD, 73

ConfigureI2CAddr  
 LCD, 73  
 ConfigureLineNumber  
 LCD, 74  
 configureTimer1  
 timer, 127  
 CpuLoad, 5  
 avg\_load, 8  
 ComputeCPUload, 6  
 CpuLoad, 6  
 current\_load, 8  
 getAverageCPUload, 7  
 getCurrentCPUload, 7  
 getMaxCPUload, 7  
 last\_sum\_value, 8  
 max\_load, 9  
 sample\_cnt, 9  
 sample\_idx, 9  
 sample\_mem, 9  
 CpuLoad.cpp, 147  
 p\_global\_BSW\_cpupload, 148  
 CpuLoad.h, 148  
 NB\_OF\_SAMPLES, 149  
 p\_global\_BSW\_cpupload, 149  
 curElement\_ptr  
 LinkedList, 89  
 current\_load  
 CpuLoad, 8  
 cursor\_en  
 T\_LCD\_conf\_struct, 112  
 cursorBlink\_en  
 T\_LCD\_conf\_struct, 112  
 DATA\_ACK  
 I2C.h, 180  
 DDRA\_PTR  
 dio\_reg\_atm2560.h, 167  
 DDRB\_PTR  
 dio\_reg\_atm2560.h, 167  
 DDRC\_PTR  
 dio\_reg\_atm2560.h, 167  
 DDRD\_PTR  
 dio\_reg\_atm2560.h, 167  
 DEBUG\_ACTIVE\_PORT  
 main.cpp, 201  
 DECODE\_PIN  
 dio.h, 162  
 DECODE\_PORT  
 dio.h, 162  
 DHT22\_PORT  
 TempSensor.h, 214  
 DISPLAY\_LINE\_SHIFT\_PERIOD\_MS  
 DisplayInterface.h, 171  
 DISPLAY\_LINE\_SHIFT\_TEMPO\_TIME  
 DisplayInterface.h, 171  
 DISPLAY\_MGT\_I2C\_BITRATE  
 DisplayManagement.h, 175  
 DISPLAY\_MGT\_LCD\_I2C\_ADDR  
 DisplayManagement.h, 175  
 DISPLAY\_MGT\_LINE\_HUM  
 DisplayManagement.h, 175  
 DISPLAY\_MGT\_LINE\_TEMP  
 DisplayManagement.h, 176  
 DISPLAY\_MGT\_PERIOD\_TASK\_SENSOR  
 DisplayManagement.h, 176  
 DISPLAY\_MGT\_PERIOD\_WELCOME\_MSG\_REMOTE\_VAL  
 DisplayManagement.h, 176  
 data\_ptr  
 LinkedList::T\_LL\_element, 114  
 ddram\_addr  
 LCD, 81  
 debug\_ift\_ptr  
 DebugManagement, 27  
 debug\_mgt\_main\_menu\_state\_t  
 DebugManagement.h, 156  
 debug\_mgt\_state\_struct\_t, 10  
 main\_state, 10  
 wdg\_state, 10  
 debug\_mgt\_wdg\_state\_t  
 DebugManagement.h, 157  
 debug\_state  
 DebugManagement, 27  
 DebugInterface, 11  
 ClearScreen, 12  
 DebugInterface, 12  
 nextLine, 12  
 read, 13  
 sendBool, 13  
 sendChar, 14  
 sendInteger, 15  
 sendString, 16, 17  
 usart\_drv\_ptr, 17  
 DebugInterface.cpp, 150  
 p\_global\_ASW\_DebugInterface, 150  
 DebugInterface.h, 151  
 p\_global\_ASW\_DebugInterface, 152  
 USART\_BAUDRATE, 151  
 DebugManagement, 18  
 debug\_ift\_ptr, 27  
 debug\_state, 27  
 DebugManagement, 19  
 DebugModeManagement, 20  
 DisplayData, 21  
 DisplayPeriodicData\_task, 22  
 exitDebugMenu, 23  
 getIftPtr, 24  
 getInfoStringPtr, 24  
 getMenuStringPtr, 24  
 info\_string\_ptr, 27  
 MainMenuManagement, 24  
 menu\_string\_ptr, 28  
 setInfoStringPtr, 25  
 systemReset, 26  
 tempSensor\_ptr, 28  
 WatchdogMenuManagement, 26  
 DebugManagement.cpp, 152

p\_global\_ASW\_DebugManagement, 153  
str\_debug\_info\_message\_empty, 153  
str\_debug\_info\_message\_wdg\_tmo\_updated, 153  
str\_debug\_info\_message\_wrong\_menu\_selection, 154  
str\_debug\_main\_menu, 154  
str\_debug\_wdg\_menu, 154  
str\_debug\_wdg\_timeout\_update\_selection, 154  
DebugManagement.h, 155  
debug\_mgt\_main\_menu\_state\_t, 156  
debug\_mgt\_wdg\_state\_t, 157  
p\_global\_ASW\_DebugManagement, 157  
PERIOD\_MS\_TASK\_DISPLAY\_CPU\_LOAD, 156  
PERIOD\_MS\_TASK\_DISPLAY\_DEBUG\_DATA, 156  
DebugModeManagement  
    DebugManagement, 20  
dht22, 28  
    dht22, 29  
    dht22\_port, 32  
    dio\_ptr, 32  
    initializeCommunication, 30  
    read, 30  
dht22.cpp, 157  
    MAX\_WAIT\_TIME\_US, 158  
    p\_global\_BSW\_dht22, 159  
dht22.h, 159  
    p\_global\_BSW\_dht22, 160  
dht22\_port  
    dht22, 32  
dio, 33  
    dio, 34  
    dio\_changePortPinCnf, 34  
    dio\_getPort, 35  
    dio\_getPort\_fast, 36  
    dio\_invertPort, 36  
    dio\_memorizePINaddress, 37  
    dio\_setPort, 38  
    getDDRxAddress, 39  
    getPINxAddress, 40  
    getPORTxAddress, 40  
    PINx\_addr\_mem, 41  
    PINx\_idx\_mem, 41  
    ports\_init, 41  
dio.cpp, 160  
    p\_global\_BSW\_dio, 161  
dio.h, 161  
    DECODE\_PIN, 162  
    DECODE\_PORT, 162  
    ENCODE\_PORT, 162  
    p\_global\_BSW\_dio, 163  
    PORT\_CNF\_IN, 163  
    PORT\_CNF\_OUT, 163  
dio\_changePortPinCnf  
    dio, 34  
dio\_getPort  
    dio, 35  
dio\_getPort\_fast  
    dio, 36  
dio\_invertPort  
    dio, 36  
dio\_memorizePINaddress  
    dio, 37  
dio\_port\_cnf.h, 164  
    PORT\_A, 164  
    PORT\_B, 164  
    PORT\_C, 165  
    PORT\_D, 165  
    PORTB\_CNF\_DDRB, 165  
    PORTB\_CNF\_PORTB, 165  
dio\_ptr  
    dht22, 32  
dio\_reg\_atm2560.h, 166  
    DDRA\_PTR, 167  
    DDRB\_PTR, 167  
    DDRC\_PTR, 167  
    DDRD\_PTR, 167  
    PINA\_PTR, 168  
    PINB\_PTR, 168  
    PINC\_PTR, 168  
    PIND\_PTR, 168  
    PORTA\_PTR, 168  
    PORTB\_PTR, 169  
    PORTC\_PTR, 169  
    PORTD\_PTR, 169  
dio\_setPort  
    dio, 38  
disable  
    Watchdog, 137  
display\_data  
    DisplayInterface, 53  
display\_en  
    T\_LCD\_conf\_struct, 112  
display\_str  
    T\_display\_data, 109  
DisplayData  
    DebugManagement, 21  
DisplayFullLine  
    DisplayInterface, 47  
DisplayInterface, 42  
    ClearFullScreen, 44  
    ClearLine, 45  
    ClearStringInDataStruct, 46  
    display\_data, 53  
    DisplayFullLine, 47  
    DisplayInterface, 44  
    dummy, 53  
    FindFirstCharAddr, 47  
    getDisplayDataPtr, 48  
    IsLineEmpty, 49  
    isShiftInProgress, 53  
    p\_lcd, 53  
    RefreshLine, 49  
    setLineAlignment, 50  
    setLineAlignmentAndRefresh, 51  
    shiftLine\_task, 51

updateLineAndRefresh, 52  
**DisplayInterface.cpp**, 169  
 p\_global\_ASW\_DisplayInterface, 170  
**DisplayInterface.h**, 170  
 DISPLAY\_LINE\_SHIFT\_PERIOD\_MS, 171  
 DISPLAY\_LINE\_SHIFT\_TEMPO\_TIME, 171  
 p\_global\_ASW\_DisplayInterface, 173  
 T\_DisplayInterface\_LineAlignment, 172  
 T\_DisplayInterface\_LineDisplayMode, 172  
**DisplayManagement**, 54  
 DisplayManagement, 55  
 DisplaySensorData\_Task, 56  
 GetIftPointer, 57  
 GetTempSensorPtr, 57  
 p\_display\_ift, 59  
 p\_tempSensor, 59  
 RemoveWelcomeMessage\_Task, 58  
**DisplayManagement.cpp**, 173  
 p\_global\_ASW\_DisplayManagement, 174  
**DisplayManagement.h**, 174  
 DISPLAY\_MGT\_I2C\_BITRATE, 175  
 DISPLAY\_MGT\_LCD\_I2C\_ADDR, 175  
 DISPLAY\_MGT\_LINE\_HUM, 175  
 DISPLAY\_MGT\_LINE\_TEMP, 176  
 DISPLAY\_MGT\_PERIOD\_TASK\_SENSOR, 176  
 DISPLAY\_MGT\_PERIOD\_WELCOME\_MSG\_R←  
     EMOVAL, 176  
 humidityDisplayString, 176  
 LCD\_init\_cnf, 176  
 noSensorDisplayString, 177  
 p\_global\_ASW\_DisplayManagement, 177  
 tempDisplayString, 177  
 welcomeMessageString, 177  
**DisplayPeriodicData\_task**  
 DebugManagement, 22  
**DisplaySensorData\_Task**  
 DisplayManagement, 56  
**dummy**  
 DisplayInterface, 53  
  
**EN\_PIN**  
 LCD.h, 189  
**ENCODE\_PORT**  
 dio.h, 162  
**enable**  
 Watchdog, 137  
**entryModeDir**  
 T\_LCD\_conf\_struct, 112  
**entryModeShift**  
 T\_LCD\_conf\_struct, 112  
**exitDebugMenu**  
 DebugManagement, 23  
  
**FindElement**  
 LinkedList, 85  
**FindFirstCharAddr**  
 DisplayInterface, 47  
**firstElement**  
 LinkedList, 89  
  
 fontType\_cnf  
 T\_LCD\_conf\_struct, 113  
  
 getAverageCPUload  
 CpuLoad, 7  
**getCurrentElement**  
 LinkedList, 86  
**getCurrentCPUload**  
 CpuLoad, 7  
**GetDDRAMAddress**  
 LCD, 75  
**getDDRxAddress**  
 dio, 39  
**getDisplayDataPtr**  
 DisplayInterface, 48  
**GetHumDecimal**  
 TempSensor, 117  
**GetHumInteger**  
 TempSensor, 118  
**getHumPtr**  
 TempSensor, 118  
**getHumidity**  
 TempSensor, 117  
**GetIftPointer**  
 DisplayManagement, 57  
**getIftPtr**  
 DebugManagement, 24  
**getInfoStringPtr**  
 DebugManagement, 24  
**GetLineNumberCnf**  
 LCD, 75  
**getMaxCPUload**  
 CpuLoad, 7  
**getMenuStringPtr**  
 DebugManagement, 24  
**getPINxAddress**  
 dio, 40  
**getPORTxAddress**  
 dio, 40  
**getPitNumber**  
 scheduler, 92  
**getSize**  
 String, 104  
**getString**  
 String, 105  
**getTaskCount**  
 scheduler, 92  
**getTaskPeriod**  
 TempSensor, 119  
**getTemp**  
 TempSensor, 119  
**GetTempDecimal**  
 TempSensor, 120  
**GetTempInteger**  
 TempSensor, 120  
**getTempPtr**  
 TempSensor, 120  
**GetTempSensorPtr**  
 DisplayManagement, 57

getTimer1Value  
    timer, 128  
GetValidity  
    TempSensor, 121  
  
humidityDisplayString  
    DisplayManagement.h, 176  
  
I2C.cpp, 178  
    p\_global\_BSW\_i2c, 178  
I2C.h, 179  
    DATA\_ACK, 180  
    p\_global\_BSW\_i2c, 180  
    SLA\_ACK, 180  
    START, 180  
I2C, 59  
    bitrate, 63  
    I2C, 60  
    initializeBus, 61  
    setBitRate, 61  
    setTxAddress, 62  
    tx\_address, 63  
    writeByte, 62  
i2c\_addr  
    T\_LCD\_conf\_struct, 113  
i2c\_bitrate  
    T\_LCD\_conf\_struct, 113  
i2c\_drv\_ptr  
    LCD, 81  
ISR  
    int.cpp, 181, 182  
info\_string\_ptr  
    DebugManagement, 27  
initializeBus  
    I2C, 61  
initializeCommunication  
    dht22, 30  
InitializeScreen  
    LCD, 75  
int.cpp, 181  
    ISR, 181, 182  
isDebugEnabled  
    T\_ASW\_init\_cnf, 107  
isDebugModeActivated  
    main.cpp, 202  
    main.h, 203  
isDisplayActivated  
    T\_ASW\_init\_cnf, 107  
isEmpty  
    T\_display\_data, 109  
isLEDActivated  
    T\_ASW\_init\_cnf, 107  
IsLLEmpty  
    LinkedList, 86  
IsLineEmpty  
    DisplayInterface, 49  
isShiftInProgress  
    DisplayInterface, 53  
isTempSensorActivated  
    T\_ASW\_init\_cnf, 107  
keepAliveLed, 63  
    blinkLed\_task, 64  
    keepAliveLed, 64  
keepAliveLed.cpp, 183  
    p\_global\_ASW\_keepAliveLed, 184  
keepAliveLed.h, 184  
    LED\_PORT, 185  
    p\_global\_ASW\_keepAliveLed, 185  
    PERIOD\_MS\_TASK\_LED, 185  
  
LCD.cpp, 186  
    p\_global\_BSW\_lcd, 186  
LCD.h, 187  
    BACKLIGHT\_PIN, 188  
    EN\_PIN, 189  
    LCD\_CNF\_BACKLIGHT\_OFF, 189  
    LCD\_CNF\_BACKLIGHT\_ON, 189  
    LCD\_CNF\_CURSOR\_BLINK\_OFF, 189  
    LCD\_CNF\_CURSOR\_BLINK\_ON, 189  
    LCD\_CNF\_CURSOR\_OFF, 190  
    LCD\_CNF\_CURSOR\_ON, 190  
    LCD\_CNF\_DISPLAY\_OFF, 190  
    LCD\_CNF\_DISPLAY\_ON, 190  
    LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_LEFT, 190  
    LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_RIGHT, 191  
    LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT←OFF, 191  
    LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT←ON, 191  
    LCD\_CNF\_FONT\_5\_11, 191  
    LCD\_CNF\_FONT\_5\_8, 191  
    LCD\_CNF\_ONE\_LINE, 192  
    LCD\_CNF\_SHIFT\_ID, 192  
    LCD\_CNF\_SHIFT\_SH, 192  
    LCD\_CNF\_TWO\_LINE, 192  
    LCD\_DISPLAY\_CTRL\_FIELD\_B, 192  
    LCD\_DISPLAY\_CTRL\_FIELD\_C, 193  
    LCD\_DISPLAY\_CTRL\_FIELD\_D, 193  
    LCD\_FCT\_SET\_FIELD\_DL, 193  
    LCD\_FCT\_SET\_FIELD\_F, 193  
    LCD\_FCT\_SET\_FIELD\_N, 193  
    LCD\_INST\_CLR\_DISPLAY\_BIT, 194  
    LCD\_INST\_DISPLAY\_CTRL, 194  
    LCD\_INST\_ENTRY\_MODE\_SET, 194  
    LCD\_INST\_FUNCTION\_SET, 194  
    LCD\_INST\_SET\_DDRAM\_ADDR, 194  
    LCD\_RAM\_1\_LINE\_MAX, 195  
    LCD\_RAM\_1\_LINE\_MIN, 195  
    LCD\_RAM\_2\_LINES\_MAX\_1, 195  
    LCD\_RAM\_2\_LINES\_MAX\_2, 195  
    LCD\_RAM\_2\_LINES\_MIN\_1, 195  
    LCD\_RAM\_2\_LINES\_MIN\_2, 196  
    LCD\_SIZE\_NB\_CHAR\_PER\_LINE, 196  
    LCD\_SIZE\_NB\_LINES, 196  
    LCD\_WAIT\_CLR\_RETURN, 196

LCD\_WAIT\_OTHER\_MODES, 196  
 p\_global\_BSW\_lcd, 198  
 RS\_PIN, 197  
 RW\_PIN, 197  
 T\_LCD\_command, 197  
 T\_LCD\_config\_mode, 198  
 T\_LCD\_ram\_area, 198  
 LCD\_CNF\_BACKLIGHT\_OFF  
     LCD.h, 189  
 LCD\_CNF\_BACKLIGHT\_ON  
     LCD.h, 189  
 LCD\_CNF\_CURSOR\_BLINK\_OFF  
     LCD.h, 189  
 LCD\_CNF\_CURSOR\_BLINK\_ON  
     LCD.h, 189  
 LCD\_CNF\_CURSOR\_OFF  
     LCD.h, 190  
 LCD\_CNF\_CURSOR\_ON  
     LCD.h, 190  
 LCD\_CNF\_DISPLAY\_OFF  
     LCD.h, 190  
 LCD\_CNF\_DISPLAY\_ON  
     LCD.h, 190  
 LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_LEFT  
     LCD.h, 190  
 LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_RIGHT  
     LCD.h, 191  
 LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT\_OFF  
     LCD.h, 191  
 LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT\_ON  
     LCD.h, 191  
 LCD\_CNF\_FONT\_5\_11  
     LCD.h, 191  
 LCD\_CNF\_FONT\_5\_8  
     LCD.h, 191  
 LCD\_CNF\_ONE\_LINE  
     LCD.h, 192  
 LCD\_CNF\_SHIFT\_ID  
     LCD.h, 192  
 LCD\_CNF\_SHIFT\_SH  
     LCD.h, 192  
 LCD\_CNF\_TWO\_LINE  
     LCD.h, 192  
 LCD\_DISPLAY\_CTRL\_FIELD\_B  
     LCD.h, 192  
 LCD\_DISPLAY\_CTRL\_FIELD\_C  
     LCD.h, 193  
 LCD\_DISPLAY\_CTRL\_FIELD\_D  
     LCD.h, 193  
 LCD\_FCT\_SET\_FIELD\_DL  
     LCD.h, 193  
 LCD\_FCT\_SET\_FIELD\_F  
     LCD.h, 193  
 LCD\_FCT\_SET\_FIELD\_N  
     LCD.h, 193  
 LCD\_INST\_CLR\_DISPLAY\_BIT  
     LCD.h, 194  
 LCD\_INST\_DISPLAY\_CTRL  
     LCD.h, 194  
     LCD.h, 194  
 LCD\_INST\_ENTRY\_MODE\_SET  
     LCD.h, 194  
 LCD\_INST\_FUNCTION\_SET  
     LCD.h, 194  
 LCD\_INST\_SET\_DDRAM\_ADDR  
     LCD.h, 194  
 LCD\_RAM\_1\_LINE\_MAX  
     LCD.h, 195  
 LCD\_RAM\_1\_LINE\_MIN  
     LCD.h, 195  
 LCD\_RAM\_2\_LINES\_MAX\_1  
     LCD.h, 195  
 LCD\_RAM\_2\_LINES\_MAX\_2  
     LCD.h, 195  
 LCD\_RAM\_2\_LINES\_MIN\_1  
     LCD.h, 195  
 LCD\_RAM\_2\_LINES\_MIN\_2  
     LCD.h, 196  
 LCD\_SIZE\_NB\_CHAR\_PER\_LINE  
     LCD.h, 196  
 LCD\_SIZE\_NB\_LINES  
     LCD.h, 196  
 LCD\_WAIT\_CLR\_RETURN  
     LCD.h, 196  
 LCD\_WAIT\_OTHER\_MODES  
     LCD.h, 196  
 LCD\_init\_cnf  
     DisplayManagement.h, 176  
 LCD, 65  
     backlight\_enable, 79  
     cnfCursorBlink, 80  
     cnfCursorOnOff, 80  
     cnfDisplayOnOff, 80  
     cnfEntryModeDir, 80  
     cnfEntryModeShift, 80  
     cnfFontType, 81  
     cnfI2C\_addr, 81  
     cnfLineNumber, 81  
     command, 68  
     ConfigureBacklight, 69  
     ConfigureCursorBlink, 69  
     ConfigureCursorOnOff, 70  
     ConfigureDisplayOnOff, 71  
     ConfigureEntryModeDir, 71  
     ConfigureEntryModeShift, 72  
     ConfigureFontType, 73  
     ConfigureI2CAddr, 73  
     ConfigureLineNumber, 74  
     ddram\_addr, 81  
     GetDDRAMAddress, 75  
     GetLineNumberCnf, 75  
     i2c\_drv\_ptr, 81  
     InitializeScreen, 75  
     LCD, 67  
     SetDDRAMAddress, 76  
     write, 77  
     write4bits, 78

WriteInRam, 79  
LED\_PORT  
    keepAliveLed.h, 185  
LLElementCompare  
    scheduler, 93  
last\_sum\_value  
    CpuLoad, 8  
launchPeriodicTasks  
    scheduler, 93  
lineNumber\_cnf  
    T\_LCD\_conf\_struct, 113  
LinkedList, 82  
    ~LinkedList, 84  
    AttachNewElement, 85  
    curElement\_ptr, 89  
    FindElement, 85  
    firstElement, 89  
    getCurrentElement, 86  
    IsLLEmpty, 86  
    LinkedList, 84  
    MoveToNextElement, 87  
    RemoveElement, 87  
    ResetElementPtr, 88  
    T\_LL\_element, 83  
LinkedList.cpp, 199  
LinkedList.h, 199  
    CompareFctPtr\_t, 200  
LinkedList::T\_LL\_element, 114  
    data\_ptr, 114  
    nextElement, 114  
  
MAX\_WAIT\_TIME\_US  
    dht22.cpp, 158  
main  
    main.cpp, 201  
main.cpp, 200  
    ASW\_init\_cnf, 202  
    DEBUG\_ACTIVE\_PORT, 201  
    isDebugModeActivated, 202  
    main, 201  
main.h, 203  
    ASW\_init\_cnf, 203  
    isDebugModeActivated, 203  
main\_state  
    debug\_mgt\_state\_struct\_t, 10  
MainMenuManagement  
    DebugManagement, 24  
max\_load  
    CpuLoad, 9  
menu\_string\_ptr  
    DebugManagement, 28  
mode  
    T\_display\_data, 109  
MoveToNextElement  
    LinkedList, 87  
  
NB\_OF\_SAMPLES  
    CpuLoad.h, 149  
nextElement  
    operator delete, 205  
    operators.h, 206  
    operator new, 205  
    operators.h, 207  
    operator delete, 205  
    operator new, 205  
    operators.h, 205  
    operator delete, 206  
    operator new, 207  
  
p\_display\_ift  
    DisplayManagement, 59  
p\_global\_ASW\_DebugInterface  
    DebugInterface.cpp, 150  
    DebugInterface.h, 152  
p\_global\_ASW\_DebugManagement  
    DebugManagement.cpp, 153  
    DebugManagement.h, 157  
p\_global\_ASW\_DisplayInterface  
    DisplayInterface.cpp, 170  
    DisplayInterface.h, 173  
p\_global\_ASW\_DisplayManagement  
    DisplayManagement.cpp, 174  
    DisplayManagement.h, 177  
p\_global\_ASW\_TempSensor  
    TempSensor.cpp, 213  
    TempSensor.h, 214  
p\_global\_ASW\_KeepAliveLed  
    keepAliveLed.cpp, 184  
    keepAliveLed.h, 185  
p\_global\_BSW\_cupload  
    CpuLoad.cpp, 148  
    CpuLoad.h, 149  
p\_global\_BSW\_dht22  
    dht22.cpp, 159  
    dht22.h, 160  
p\_global\_BSW\_dio  
    dio.cpp, 161  
    dio.h, 163  
p\_global\_BSW\_i2c  
    I2C.cpp, 178  
    I2C.h, 180  
p\_global\_BSW\_lcd  
    LCD.cpp, 186  
    LCD.h, 198  
p\_global\_BSW\_timer  
    timer.cpp, 215  
    timer.h, 216  
p\_global\_BSW\_usart  
    USART.cpp, 217

usart.h, 218  
**p\_global\_BSW\_wdg**  
 Watchdog.cpp, 220  
 Watchdog.h, 223  
**p\_global\_scheduler**  
 scheduler.cpp, 208  
 scheduler.h, 210  
**p\_lcd**  
 DisplayInterface, 53  
**p\_tempSensor**  
 DisplayManagement, 59  
**PERIOD\_MS\_TASK\_DISPLAY\_CPU\_LOAD**  
 DebugManagement.h, 156  
**PERIOD\_MS\_TASK\_DISPLAY\_DEBUG\_DATA**  
 DebugManagement.h, 156  
**PERIOD\_MS\_TASK\_LED**  
 keepAliveLed.h, 185  
**PERIOD\_MS\_TASK\_TEMP\_SENSOR**  
 TempSensor.h, 214  
**PINA\_PTR**  
 dio\_reg\_atm2560.h, 168  
**PINB\_PTR**  
 dio\_reg\_atm2560.h, 168  
**PINC\_PTR**  
 dio\_reg\_atm2560.h, 168  
**PIND\_PTR**  
 dio\_reg\_atm2560.h, 168  
**PINx\_addr\_mem**  
 dio, 41  
**PINx\_idx\_mem**  
 dio, 41  
**PIT\_BEFORE\_INVALID**  
 TempSensor.cpp, 212  
**PORT\_CNF\_IN**  
 dio.h, 163  
**PORT\_CNF\_OUT**  
 dio.h, 163  
**PORT\_A**  
 dio\_port\_cnf.h, 164  
**PORT\_B**  
 dio\_port\_cnf.h, 164  
**PORT\_C**  
 dio\_port\_cnf.h, 165  
**PORT\_D**  
 dio\_port\_cnf.h, 165  
**PORTA\_PTR**  
 dio\_reg\_atm2560.h, 168  
**PORTB\_CNF\_DDRB**  
 dio\_port\_cnf.h, 165  
**PORTB\_CNF\_PORTB**  
 dio\_port\_cnf.h, 165  
**PORTB\_PTR**  
 dio\_reg\_atm2560.h, 169  
**PORTC\_PTR**  
 dio\_reg\_atm2560.h, 169  
**PORTD\_PTR**  
 dio\_reg\_atm2560.h, 169  
**PRESCALER\_PERIODIC\_TIMER**  
 scheduler.h, 209  
**period**  
 scheduler::Task\_t, 115  
**pit\_number**  
 scheduler, 97  
**ports\_init**  
 dio, 41  
**prescaler**  
 timer, 129  
**RS\_PIN**  
 LCD.h, 197  
**RW\_PIN**  
 LCD.h, 197  
**read**  
 DebugInterface, 13  
 dht22, 30  
**read\_humidity**  
 TempSensor, 124  
**read\_temperature**  
 TempSensor, 124  
**readTempSensor\_task**  
 TempSensor, 121  
**RefreshLine**  
 DisplayInterface, 49  
**RemoveElement**  
 LinkedList, 87  
**removePeriodicTask**  
 scheduler, 94  
**RemoveWelcomeMessage\_Task**  
 DisplayManagement, 58  
**reset**  
 Watchdog, 138  
**ResetElementPtr**  
 LinkedList, 88  
**SLA\_ACK**  
 I2C.h, 180  
**START**  
 I2C.h, 180  
**SW\_PERIOD\_MS**  
 scheduler.h, 209  
**sample\_cnt**  
 CpuLoad, 9  
**sample\_idx**  
 CpuLoad, 9  
**sample\_mem**  
 CpuLoad, 9  
**scheduler**, 89  
 addPeriodicTask, 91  
 getPitNumber, 92  
 getTaskCount, 92  
 LL\_ElementCompare, 93  
 launchPeriodicTasks, 93  
 pit\_number, 97  
 removePeriodicTask, 94  
 scheduler, 91  
 startScheduling, 95  
 task\_count, 97

Task\_t, 91  
TasksLL\_ptr, 97  
updateTaskPeriod, 96  
scheduler.cpp, 207  
  p\_global\_scheduler, 208  
scheduler.h, 208  
  p\_global\_scheduler, 210  
PRESCALER\_PERIODIC\_TIMER, 209  
SW\_PERIOD\_MS, 209  
TIMER\_CTC\_VALUE, 209  
TaskPtr\_t, 210  
scheduler::Task\_t, 115  
  period, 115  
  TaskPtr, 115  
sendBool  
  DebugInterface, 13  
sendChar  
  DebugInterface, 14  
sendInteger  
  DebugInterface, 15  
sendString  
  DebugInterface, 16, 17  
setBaudRate  
  usart, 131  
setBitRate  
  I2C, 61  
SetDDRAMAddress  
  LCD, 76  
setInfoStringPtr  
  DebugManagement, 25  
setLineAlignment  
  DisplayInterface, 50  
setLineAlignmentAndRefresh  
  DisplayInterface, 51  
setTxAddress  
  I2C, 62  
setValidity  
  TempSensor, 122  
shift\_data  
  T\_display\_data, 109  
shiftLine\_task  
  DisplayInterface, 51  
size  
  String, 106  
startScheduling  
  scheduler, 95  
startTimer1  
  timer, 128  
stopTimer1  
  timer, 129  
str\_cur\_ptr  
  T\_Display\_shift\_data, 110  
str\_debug\_info\_message\_empty  
  DebugManagement.cpp, 153  
str\_debug\_info\_message\_wdg\_tmo\_updated  
  DebugManagement.cpp, 153  
str\_debug\_info\_message\_wrong\_menu\_selection  
  DebugManagement.cpp, 154

str\_debug\_main\_menu  
  DebugManagement.cpp, 154  
str\_debug\_wdg\_menu  
  DebugManagement.cpp, 154  
str\_debug\_wdg\_timeout\_update\_selection  
  DebugManagement.cpp, 154  
str\_ptr  
  T\_Display\_shift\_data, 110  
String, 98  
  ~String, 99  
  appendBool, 100  
  appendChar, 101  
  appendInteger, 102  
  appendString, 102  
  Clear, 103  
  ComputeStringSize, 104  
  getSize, 104  
  getString, 105  
  size, 106  
  String, 99  
  string, 106  
string  
  String, 106  
String.cpp, 210  
String.h, 211  
SystemReset  
  Watchdog, 138  
systemReset  
  DebugManagement, 26

T\_ASW\_init\_cnf, 106  
  isDebugActivated, 107  
  isDisplayActivated, 107  
  isLEDActivated, 107  
  isTempSensorActivated, 107  
T\_Display\_shift\_data, 110  
  str\_cur\_ptr, 110  
  str\_ptr, 110  
  temporization, 111  
T\_DisplayInterface\_LineAlignment  
  DisplayInterface.h, 172  
T\_DisplayInterface\_LineDisplayMode  
  DisplayInterface.h, 172  
T\_LCD\_command  
  LCD.h, 197  
T\_LCD\_conf\_struct, 111  
  backlight\_en, 112  
  cursor\_en, 112  
  cursorBlink\_en, 112  
  display\_en, 112  
  entryModeDir, 112  
  entryModeShift, 112  
  fontType\_cnf, 113  
  i2c\_addr, 113  
  i2c\_bitrate, 113  
  lineNumber\_cnf, 113  
T\_LCD\_config\_mode  
  LCD.h, 198  
T\_LCD\_ram\_area

LCD.h, 198  
 T\_LL\_element  
     LinkedList, 83  
 T\_display\_data, 108  
     alignment, 108  
     display\_str, 109  
     isEmpty, 109  
     mode, 109  
     shift\_data, 109  
 TIMER\_CTC\_VALUE  
     scheduler.h, 209  
 task\_count  
     scheduler, 97  
 task\_period  
     TempSensor, 125  
 Task\_t  
     scheduler, 91  
 TaskPtr  
     scheduler::Task\_t, 115  
 TaskPtr\_t  
     scheduler.h, 210  
 TasksLL\_ptr  
     scheduler, 97  
 tempDisplayString  
     DisplayManagement.h, 177  
 TempSensor, 115  
     GetHumDecimal, 117  
     GetHumInteger, 118  
     getHumPtr, 118  
     getHumidity, 117  
     getTaskPeriod, 119  
     getTemp, 119  
     GetTempDecimal, 120  
     GetTempInteger, 120  
     getTempPtr, 120  
     GetValidity, 121  
     read\_humidity, 124  
     read\_temperature, 124  
     readTempSensor\_task, 121  
     setValidity, 122  
     task\_period, 125  
     TempSensor, 117  
     updateLastValidValues, 123  
     updateTaskPeriod, 123  
     valid\_hum, 125  
     valid坑, 125  
     valid\_temp, 125  
     validity, 125  
     validity\_last\_read, 126  
 TempSensor.cpp, 212  
     p\_global\_ASW\_TempSensor, 213  
     PIT\_BEFORE\_INVALID, 212  
 TempSensor.h, 213  
     DHT22\_PORT, 214  
     p\_global\_ASW\_TempSensor, 214  
     PERIOD\_MS\_TASK\_TEMP\_SENSOR, 214  
 tempSensor\_ptr  
     DebugManagement, 28  
 temporization  
     T\_Display\_shift\_data, 111  
 timeoutUpdate  
     Watchdog, 139  
 timer, 126  
     configureTimer1, 127  
     getTimer1Value, 128  
     prescaler, 129  
     startTimer1, 128  
     stopTimer1, 129  
     timer, 127  
 timer.cpp, 215  
     p\_global\_BSW\_timer, 215  
 timer.h, 216  
     p\_global\_BSW\_timer, 216  
 tx\_address  
     I2C, 63  
 USART\_BAUDRATE  
     DebugInterface.h, 151  
 updateLastValidValues  
     TempSensor, 123  
 updateLineAndRefresh  
     DisplayInterface, 52  
 updateTaskPeriod  
     scheduler, 96  
     TempSensor, 123  
 usart, 130  
     BaudRate, 135  
     setBaudRate, 131  
     usart, 130  
     usart\_init, 131  
     usart\_read, 132  
     usart\_sendByte, 132  
     usart\_sendString, 133  
     usart\_transmit, 134  
 usart.cpp, 217  
     p\_global\_BSW\_usart, 217  
 usart.h, 218  
     p\_global\_BSW\_usart, 218  
 usart\_drv\_ptr  
     DebugInterface, 17  
 usart\_init  
     usart, 131  
 usart\_read  
     usart, 132  
 usart\_sendByte  
     usart, 132  
 usart\_sendString  
     usart, 133  
 usart\_transmit  
     usart, 134  
 valid\_hum  
     TempSensor, 125  
 valid坑  
     TempSensor, 125  
 valid\_temp  
     TempSensor, 125

validity  
    TempSensor, 125  
validity\_last\_read  
    TempSensor, 126

WDG\_TIMEOUT\_DEFAULT\_MS  
    Watchdog.cpp, 219

WDG\_TMO\_120MS  
    Watchdog.h, 221

WDG\_TMO\_15MS  
    Watchdog.h, 221

WDG\_TMO\_1S  
    Watchdog.h, 222

WDG\_TMO\_250MS  
    Watchdog.h, 222

WDG\_TMO\_2S  
    Watchdog.h, 222

WDG\_TMO\_30MS  
    Watchdog.h, 222

WDG\_TMO\_4S  
    Watchdog.h, 222

WDG\_TMO\_500MS  
    Watchdog.h, 223

WDG\_TMO\_60MS  
    Watchdog.h, 223

WDG\_TMO\_8S  
    Watchdog.h, 223

Watchdog, 135  
    disable, 137  
    enable, 137  
    reset, 138  
    SystemReset, 138  
    timeoutUpdate, 139  
    Watchdog, 136

Watchdog.cpp, 219  
    p\_global\_BSW\_wdg, 220  
    WDG\_TIMEOUT\_DEFAULT\_MS, 219

Watchdog.h, 220  
    p\_global\_BSW\_wdg, 223  
    WDG\_TMO\_120MS, 221  
    WDG\_TMO\_15MS, 221  
    WDG\_TMO\_1S, 222  
    WDG\_TMO\_250MS, 222  
    WDG\_TMO\_2S, 222  
    WDG\_TMO\_30MS, 222  
    WDG\_TMO\_4S, 222  
    WDG\_TMO\_500MS, 223  
    WDG\_TMO\_60MS, 223  
    WDG\_TMO\_8S, 223

WatchdogMenuManagement  
    DebugManagement, 26

wdg\_state  
    debug\_mgt\_state\_struct\_t, 10

welcomeMessageString  
    DisplayManagement.h, 177

write  
    LCD, 77

write4bits  
    LCD, 78