

Arduino

1.0

Generated by Doxygen 1.8.14



# Contents

<b>1</b>	<b>Hierarchical Index</b>	<b>1</b>
1.1	Class Hierarchy . . . . .	1
<b>2</b>	<b>Class Index</b>	<b>3</b>
2.1	Class List . . . . .	3
<b>3</b>	<b>File Index</b>	<b>5</b>
3.1	File List . . . . .	5
<b>4</b>	<b>Class Documentation</b>	<b>9</b>
4.1	CpuLoad Class Reference . . . . .	9
4.1.1	Detailed Description . . . . .	9
4.1.2	Constructor & Destructor Documentation . . . . .	10
4.1.2.1	CpuLoad() . . . . .	10
4.1.3	Member Function Documentation . . . . .	10
4.1.3.1	ComputeCPULoad() . . . . .	10
4.1.3.2	getAverageCPULoad() . . . . .	11
4.1.3.3	getCurrentCPULoad() . . . . .	11
4.1.3.4	getMaxCPULoad() . . . . .	12
4.1.4	Member Data Documentation . . . . .	12
4.1.4.1	avg_load . . . . .	12
4.1.4.2	current_load . . . . .	12
4.1.4.3	last_sum_value . . . . .	13
4.1.4.4	max_load . . . . .	13
4.1.4.5	sample_cnt . . . . .	13

4.1.4.6	sample_idx . . . . .	13
4.1.4.7	sample_mem . . . . .	13
4.2	debug_mgt_state_struct_t Struct Reference . . . . .	14
4.2.1	Detailed Description . . . . .	14
4.2.2	Member Data Documentation . . . . .	14
4.2.2.1	main_state . . . . .	14
4.2.2.2	wdg_state . . . . .	14
4.3	DebugInterface Class Reference . . . . .	15
4.3.1	Detailed Description . . . . .	16
4.3.2	Constructor & Destructor Documentation . . . . .	16
4.3.2.1	DebugInterface() . . . . .	16
4.3.3	Member Function Documentation . . . . .	16
4.3.3.1	ClearScreen() . . . . .	16
4.3.3.2	nextLine() . . . . .	17
4.3.3.3	read() . . . . .	17
4.3.3.4	sendBool() . . . . .	18
4.3.3.5	sendChar() . . . . .	18
4.3.3.6	sendInteger() . . . . .	19
4.3.3.7	sendString() [1/2] . . . . .	20
4.3.3.8	sendString() [2/2] . . . . .	21
4.3.4	Member Data Documentation . . . . .	21
4.3.4.1	uart_drv_ptr . . . . .	22
4.4	DebugManagement Class Reference . . . . .	22
4.4.1	Detailed Description . . . . .	23
4.4.2	Constructor & Destructor Documentation . . . . .	23
4.4.2.1	DebugManagement() . . . . .	24
4.4.3	Member Function Documentation . . . . .	24
4.4.3.1	DebugModeManagement() . . . . .	24
4.4.3.2	DisplayData() . . . . .	25
4.4.3.3	DisplayPeriodicData_task() . . . . .	26

4.4.3.4	exitDebugMenu()	27
4.4.3.5	getIftPtr()	28
4.4.3.6	getInfoStringPtr()	28
4.4.3.7	getMenuStringPtr()	29
4.4.3.8	MainMenuManagement()	29
4.4.3.9	setInfoStringPtr()	30
4.4.3.10	systemReset()	30
4.4.3.11	WatchdogMenuManagement()	31
4.4.4	Member Data Documentation	32
4.4.4.1	debug_ift_ptr	32
4.4.4.2	debug_state	32
4.4.4.3	info_string_ptr	33
4.4.4.4	isInfoStringDisplayed	33
4.4.4.5	menu_string_ptr	33
4.4.4.6	sensorMgt_ptr	33
4.5	dht22 Class Reference	34
4.5.1	Detailed Description	35
4.5.2	Constructor & Destructor Documentation	35
4.5.2.1	dht22()	35
4.5.3	Member Function Documentation	35
4.5.3.1	getHumidity()	36
4.5.3.2	getTemperature()	37
4.5.3.3	initializeCommunication()	38
4.5.3.4	read()	39
4.5.4	Member Data Documentation	40
4.5.4.1	dht22_port	40
4.5.4.2	dio_ptr	40
4.5.4.3	mem_humidity	40
4.5.4.4	mem_temperature	40
4.5.4.5	mem_validity	40

4.5.4.6	<code>pit_last_read</code>	41
4.6	<code>dio</code> Class Reference	41
4.6.1	Detailed Description	42
4.6.2	Constructor & Destructor Documentation	42
4.6.2.1	<code>dio()</code>	42
4.6.3	Member Function Documentation	42
4.6.3.1	<code>dio_changePortPinCnf()</code>	42
4.6.3.2	<code>dio_getPort()</code>	43
4.6.3.3	<code>dio_getPort_fast()</code>	44
4.6.3.4	<code>dio_invertPort()</code>	45
4.6.3.5	<code>dio_memorizePINaddress()</code>	45
4.6.3.6	<code>dio_setPort()</code>	46
4.6.3.7	<code>getDDRxAddress()</code>	47
4.6.3.8	<code>getPINxAddress()</code>	48
4.6.3.9	<code>getPORTxAddress()</code>	48
4.6.3.10	<code>ports_init()</code>	49
4.6.4	Member Data Documentation	49
4.6.4.1	<code>PINx_addr_mem</code>	50
4.6.4.2	<code>PINx_idx_mem</code>	50
4.7	<code>DisplayInterface</code> Class Reference	50
4.7.1	Detailed Description	52
4.7.2	Constructor & Destructor Documentation	52
4.7.2.1	<code>DisplayInterface()</code>	52
4.7.3	Member Function Documentation	52
4.7.3.1	<code>ClearFullScreen()</code>	53
4.7.3.2	<code>ClearLine()</code>	53
4.7.3.3	<code>ClearStringInDataStruct()</code>	54
4.7.3.4	<code>DisplayFullLine() [1/2]</code>	55
4.7.3.5	<code>DisplayFullLine() [2/2]</code>	56
4.7.3.6	<code>FindFirstCharAddr()</code>	57

4.7.3.7	getDisplayDataPtr()	57
4.7.3.8	IsLineEmpty()	58
4.7.3.9	RefreshLine()	58
4.7.3.10	setLineAlignment()	59
4.7.3.11	setLineAlignmentAndRefresh()	60
4.7.3.12	shiftLine_task()	60
4.7.3.13	updateLineAndRefresh()	61
4.7.4	Member Data Documentation	62
4.7.4.1	display_data	62
4.7.4.2	dummy	62
4.7.4.3	isShiftInProgress	62
4.7.4.4	p_lcd	63
4.8	DisplayManagement Class Reference	63
4.8.1	Detailed Description	64
4.8.2	Constructor & Destructor Documentation	64
4.8.2.1	DisplayManagement()	64
4.8.3	Member Function Documentation	65
4.8.3.1	DisplaySensorData_Task()	65
4.8.3.2	GetIftPointer()	66
4.8.3.3	GetSensorMgtPtr()	66
4.8.3.4	RemoveWelcomeMessage_Task()	67
4.8.4	Member Data Documentation	67
4.8.4.1	p_display_ift	67
4.8.4.2	p_SensorMgt	68
4.9	HumSensor Class Reference	68
4.9.1	Detailed Description	69
4.9.2	Constructor & Destructor Documentation	69
4.9.2.1	HumSensor() [1/2]	69
4.9.2.2	HumSensor() [2/2]	70
4.9.3	Member Function Documentation	70

---

4.9.3.1	readHumSensor_task()	70
4.9.3.2	updateTaskPeriod()	71
4.10	I2C Class Reference	72
4.10.1	Detailed Description	73
4.10.2	Constructor & Destructor Documentation	73
4.10.2.1	I2C()	73
4.10.3	Member Function Documentation	73
4.10.3.1	initializeBus()	74
4.10.3.2	setBitRate()	74
4.10.3.3	write()	75
4.10.3.4	writeByte()	75
4.10.4	Member Data Documentation	76
4.10.4.1	bitrate	76
4.11	keepAliveLed Class Reference	76
4.11.1	Detailed Description	77
4.11.2	Constructor & Destructor Documentation	77
4.11.2.1	keepAliveLed()	77
4.11.3	Member Function Documentation	77
4.11.3.1	blinkLed_task()	78
4.12	LCD Class Reference	78
4.12.1	Detailed Description	80
4.12.2	Constructor & Destructor Documentation	80
4.12.2.1	LCD()	80
4.12.3	Member Function Documentation	81
4.12.3.1	command()	81
4.12.3.2	ConfigureBacklight()	82
4.12.3.3	ConfigureCursorBlink()	83
4.12.3.4	ConfigureCursorOnOff()	83
4.12.3.5	ConfigureDisplayOnOff()	84
4.12.3.6	ConfigureEntryModeDir()	84

4.12.3.7 ConfigureEntryModeShift()	85
4.12.3.8 ConfigureFontType()	86
4.12.3.9 ConfigureI2CAddr()	87
4.12.3.10 ConfigureLineNumber()	87
4.12.3.11 GetDDRAMAddress()	88
4.12.3.12 GetLineNumberCnf()	88
4.12.3.13 InitializeScreen()	89
4.12.3.14 SetDDRAMAddress()	89
4.12.3.15 write()	90
4.12.3.16 write4bits()	91
4.12.3.17 WriteInRam()	92
4.12.4 Member Data Documentation	93
4.12.4.1 backlight_enable	93
4.12.4.2 cnfCursorBlink	93
4.12.4.3 cnfCursorOnOff	93
4.12.4.4 cnfDisplayOnOff	93
4.12.4.5 cnfEntryModeDir	93
4.12.4.6 cnfEntryModeShift	94
4.12.4.7 cnfFontType	94
4.12.4.8 cnfI2C_addr	94
4.12.4.9 cnfLineNumber	94
4.12.4.10 ddram_addr	94
4.12.4.11 i2c_drv_ptr	95
4.13 LinkedList Class Reference	95
4.13.1 Detailed Description	96
4.13.2 Member Typedef Documentation	96
4.13.2.1 T_LL_element	97
4.13.3 Constructor & Destructor Documentation	97
4.13.3.1 LinkedList()	97
4.13.3.2 ~LinkedList()	97

4.13.4 Member Function Documentation . . . . .	98
4.13.4.1 AttachNewElement() . . . . .	98
4.13.4.2 FindElement() . . . . .	98
4.13.4.3 getCurrentElement() . . . . .	99
4.13.4.4 IsLLEmpty() . . . . .	100
4.13.4.5 MoveToNextElement() . . . . .	100
4.13.4.6 RemoveElement() . . . . .	101
4.13.4.7 ResetElementPtr() . . . . .	101
4.13.5 Member Data Documentation . . . . .	102
4.13.5.1 curElement_ptr . . . . .	102
4.13.5.2 firstElement . . . . .	102
4.14 scheduler Class Reference . . . . .	102
4.14.1 Detailed Description . . . . .	103
4.14.2 Member Typedef Documentation . . . . .	104
4.14.2.1 Task_t . . . . .	104
4.14.3 Constructor & Destructor Documentation . . . . .	104
4.14.3.1 scheduler() . . . . .	104
4.14.4 Member Function Documentation . . . . .	104
4.14.4.1 addPeriodicTask() . . . . .	104
4.14.4.2 getPitNumber() . . . . .	105
4.14.4.3 getTaskCount() . . . . .	106
4.14.4.4 launchPeriodicTasks() . . . . .	106
4.14.4.5 LLElementCompare() . . . . .	107
4.14.4.6 removePeriodicTask() . . . . .	108
4.14.4.7 startScheduling() . . . . .	109
4.14.4.8 updateTaskPeriod() . . . . .	110
4.14.5 Member Data Documentation . . . . .	111
4.14.5.1 pit_number . . . . .	111
4.14.5.2 task_count . . . . .	111
4.14.5.3 TasksLL_ptr . . . . .	111

4.15 Sensor Class Reference . . . . .	112
4.15.1 Detailed Description . . . . .	113
4.15.2 Constructor & Destructor Documentation . . . . .	113
4.15.2.1 Sensor() [1/2] . . . . .	113
4.15.2.2 Sensor() [2/2] . . . . .	113
4.15.3 Member Function Documentation . . . . .	114
4.15.3.1 getRawDataPtr() . . . . .	114
4.15.3.2 getTaskPeriod() . . . . .	115
4.15.3.3 getValidity() . . . . .	115
4.15.3.4 getValue() . . . . .	115
4.15.3.5 getValueDecimal() . . . . .	116
4.15.3.6 getValueInteger() . . . . .	116
4.15.3.7 readSensor_task() . . . . .	116
4.15.3.8 setLastValidity() . . . . .	116
4.15.3.9 setValidityTMO() . . . . .	117
4.15.3.10 updateTaskPeriod() . . . . .	118
4.15.3.11 updateValidData() . . . . .	118
4.15.4 Member Data Documentation . . . . .	119
4.15.4.1 raw_data . . . . .	119
4.15.4.2 task_period . . . . .	119
4.15.4.3 valid_pit . . . . .	119
4.15.4.4 valid_value . . . . .	119
4.15.4.5 validity . . . . .	119
4.15.4.6 validity_last_read . . . . .	120
4.15.4.7 validity_tmo . . . . .	120
4.16 SensorManagement Class Reference . . . . .	120
4.16.1 Detailed Description . . . . .	121
4.16.2 Constructor & Destructor Documentation . . . . .	121
4.16.2.1 SensorManagement() . . . . .	121
4.16.3 Member Function Documentation . . . . .	121

---

4.16.3.1	getFullStringFormattedValue()	121
4.16.3.2	getSensorCount()	122
4.16.3.3	getSensorObjectPtr()	123
4.16.3.4	updateTaskPeriod()	124
4.16.4	Member Data Documentation	124
4.16.4.1	nb_sensors	125
4.16.4.2	sensor_ptr_table	125
4.17	String Class Reference	125
4.17.1	Detailed Description	126
4.17.2	Constructor & Destructor Documentation	126
4.17.2.1	String() [1/2]	126
4.17.2.2	String() [2/2]	127
4.17.2.3	~String()	127
4.17.3	Member Function Documentation	128
4.17.3.1	appendBool()	128
4.17.3.2	appendChar()	129
4.17.3.3	appendInteger()	129
4.17.3.4	appendSpace()	130
4.17.3.5	appendString()	131
4.17.3.6	Clear()	132
4.17.3.7	ComputeStringSize()	133
4.17.3.8	getSize()	133
4.17.3.9	getString()	134
4.17.4	Member Data Documentation	134
4.17.4.1	size	134
4.17.4.2	string	134
4.18	T_ASW_init_cnf Struct Reference	135
4.18.1	Detailed Description	135
4.18.2	Member Data Documentation	135
4.18.2.1	isDebugActivated	135

4.18.2.2	isDisplayActivated	135
4.18.2.3	isLEDActivated	136
4.18.2.4	isSensorMgtActivated	136
4.19	T_display_data Struct Reference	136
4.19.1	Detailed Description	137
4.19.2	Member Data Documentation	137
4.19.2.1	alignment	137
4.19.2.2	display_str	137
4.19.2.3	isEmpty	137
4.19.2.4	mode	138
4.19.2.5	shift_data	138
4.20	T_Display_shift_data Struct Reference	138
4.20.1	Detailed Description	139
4.20.2	Member Data Documentation	139
4.20.2.1	str_cur_ptr	139
4.20.2.2	str_ptr	139
4.20.2.3	temporization	139
4.21	T_LCD_conf_struct Struct Reference	140
4.21.1	Detailed Description	140
4.21.2	Member Data Documentation	140
4.21.2.1	backlight_en	140
4.21.2.2	cursor_en	140
4.21.2.3	cursorBlink_en	141
4.21.2.4	display_en	141
4.21.2.5	entryModeDir	141
4.21.2.6	entryModeShift	141
4.21.2.7	fontType_cnf	141
4.21.2.8	i2c_addr	142
4.21.2.9	i2c_bitrate	142
4.21.2.10	lineNumber_cnf	142

4.22	LinkedList::T_LL_element Struct Reference . . . . .	142
4.22.1	Detailed Description . . . . .	143
4.22.2	Member Data Documentation . . . . .	143
4.22.2.1	data_ptr . . . . .	143
4.22.2.2	nextElement . . . . .	143
4.23	T_SensorManagement_Sensor_Config Struct Reference . . . . .	143
4.23.1	Detailed Description . . . . .	144
4.23.2	Member Data Documentation . . . . .	144
4.23.2.1	data_name_str . . . . .	144
4.23.2.2	period . . . . .	144
4.23.2.3	sensor_type . . . . .	144
4.23.2.4	unit_str . . . . .	144
4.23.2.5	validity_tmo . . . . .	145
4.24	scheduler::Task_t Struct Reference . . . . .	145
4.24.1	Detailed Description . . . . .	145
4.24.2	Member Data Documentation . . . . .	145
4.24.2.1	period . . . . .	145
4.24.2.2	TaskPtr . . . . .	146
4.25	TempSensor Class Reference . . . . .	146
4.25.1	Detailed Description . . . . .	147
4.25.2	Constructor & Destructor Documentation . . . . .	147
4.25.2.1	TempSensor() [1/2] . . . . .	147
4.25.2.2	TempSensor() [2/2] . . . . .	148
4.25.3	Member Function Documentation . . . . .	148
4.25.3.1	readTempSensor_task() . . . . .	148
4.25.3.2	updateTaskPeriod() . . . . .	149
4.26	timer Class Reference . . . . .	150
4.26.1	Detailed Description . . . . .	150
4.26.2	Constructor & Destructor Documentation . . . . .	151
4.26.2.1	timer() . . . . .	151

4.26.3 Member Function Documentation . . . . .	151
4.26.3.1 configureTimer1() . . . . .	151
4.26.3.2 getTimer1Value() . . . . .	152
4.26.3.3 startTimer1() . . . . .	152
4.26.3.4 stopTimer1() . . . . .	153
4.26.4 Member Data Documentation . . . . .	153
4.26.4.1 prescaler . . . . .	153
4.27 usart Class Reference . . . . .	154
4.27.1 Detailed Description . . . . .	154
4.27.2 Constructor & Destructor Documentation . . . . .	154
4.27.2.1 usart() . . . . .	154
4.27.3 Member Function Documentation . . . . .	155
4.27.3.1 setBaudRate() . . . . .	155
4.27.3.2 usart_init() . . . . .	156
4.27.3.3 usart_read() . . . . .	156
4.27.3.4 usart_sendByte() . . . . .	156
4.27.3.5 usart_sendString() . . . . .	157
4.27.3.6 usart_transmit() . . . . .	158
4.27.4 Member Data Documentation . . . . .	159
4.27.4.1 BaudRate . . . . .	159
4.28 Watchdog Class Reference . . . . .	159
4.28.1 Detailed Description . . . . .	160
4.28.2 Constructor & Destructor Documentation . . . . .	160
4.28.2.1 Watchdog() [1/2] . . . . .	160
4.28.2.2 Watchdog() [2/2] . . . . .	161
4.28.3 Member Function Documentation . . . . .	161
4.28.3.1 disable() . . . . .	161
4.28.3.2 enable() . . . . .	162
4.28.3.3 getTMOValue() . . . . .	163
4.28.3.4 isEnabled() . . . . .	163
4.28.3.5 reset() . . . . .	164
4.28.3.6 SwitchWdg() . . . . .	164
4.28.3.7 SystemReset() . . . . .	165
4.28.3.8 timeoutUpdate() . . . . .	166
4.28.4 Member Data Documentation . . . . .	166
4.28.4.1 isActive . . . . .	166
4.28.4.2 tmo_value . . . . .	167

<b>5 File Documentation</b>	<b>169</b>
5.1 asw.cpp File Reference . . . . .	169
5.1.1 Detailed Description . . . . .	170
5.1.2 Function Documentation . . . . .	170
5.1.2.1 asw_init() . . . . .	170
5.2 asw.h File Reference . . . . .	171
5.2.1 Detailed Description . . . . .	171
5.2.2 Function Documentation . . . . .	171
5.2.2.1 asw_init() . . . . .	172
5.3 bsw.cpp File Reference . . . . .	172
5.3.1 Detailed Description . . . . .	173
5.3.2 Function Documentation . . . . .	173
5.3.2.1 bsw_init() . . . . .	173
5.4 bsw.h File Reference . . . . .	174
5.4.1 Detailed Description . . . . .	174
5.4.2 Function Documentation . . . . .	174
5.4.2.1 bsw_init() . . . . .	175
5.5 CpuLoad.cpp File Reference . . . . .	175
5.5.1 Detailed Description . . . . .	176
5.5.2 Variable Documentation . . . . .	176
5.5.2.1 p_global_BSW_cpupload . . . . .	176
5.6 CpuLoad.h File Reference . . . . .	176
5.6.1 Detailed Description . . . . .	177
5.6.2 Macro Definition Documentation . . . . .	177
5.6.2.1 NB_OF_SAMPLES . . . . .	177
5.6.3 Variable Documentation . . . . .	177
5.6.3.1 p_global_BSW_cpupload . . . . .	177
5.7 DebugInterface.cpp File Reference . . . . .	178
5.7.1 Detailed Description . . . . .	178
5.7.2 Variable Documentation . . . . .	178

5.7.2.1	p_global_ASW_DebugInterface	178
5.8	DebugInterface.h File Reference	179
5.8.1	Detailed Description	179
5.8.2	Macro Definition Documentation	179
5.8.2.1	USART_BAUDRATE	180
5.8.3	Variable Documentation	180
5.8.3.1	p_global_ASW_DebugInterface	180
5.9	DebugManagement.cpp File Reference	180
5.9.1	Detailed Description	181
5.9.2	Variable Documentation	181
5.9.2.1	p_global_ASW_DebugManagement	181
5.9.2.2	str_debug_info_message_wdg_disabled	182
5.9.2.3	str_debug_info_message_wdg_enabled	182
5.9.2.4	str_debug_info_message_wdg_tmo_updated	182
5.9.2.5	str_debug_info_message_wdg_tmo_value	182
5.9.2.6	str_debug_info_message_wrong_menu_selection	182
5.9.2.7	str_debug_main_menu	183
5.9.2.8	str_debug_wdg_menu	183
5.9.2.9	str_debug_wdg_timeout_update_selection	183
5.10	DebugManagement.h File Reference	184
5.10.1	Detailed Description	184
5.10.2	Macro Definition Documentation	185
5.10.2.1	PERIOD_MS_TASK_DISPLAY_CPU_LOAD	185
5.10.2.2	PERIOD_MS_TASK_DISPLAY_DEBUG_DATA	185
5.10.3	Enumeration Type Documentation	185
5.10.3.1	debug_mgt_main_menu_state_t	185
5.10.3.2	debug_mgt_wdg_state_t	185
5.10.4	Variable Documentation	186
5.10.4.1	p_global_ASW_DebugManagement	186
5.11	dht22.cpp File Reference	186

5.11.1 Detailed Description . . . . .	187
5.11.2 Macro Definition Documentation . . . . .	187
5.11.2.1 MAX_WAIT_TIME_US . . . . .	187
5.11.3 Variable Documentation . . . . .	187
5.11.3.1 p_global_BSW_dht22 . . . . .	187
5.12 dht22.h File Reference . . . . .	188
5.12.1 Detailed Description . . . . .	188
5.12.2 Variable Documentation . . . . .	188
5.12.2.1 p_global_BSW_dht22 . . . . .	188
5.13 dio.cpp File Reference . . . . .	189
5.13.1 Detailed Description . . . . .	189
5.13.2 Variable Documentation . . . . .	189
5.13.2.1 p_global_BSW_dio . . . . .	190
5.14 dio.h File Reference . . . . .	190
5.14.1 Detailed Description . . . . .	191
5.14.2 Macro Definition Documentation . . . . .	191
5.14.2.1 DECODE_PIN . . . . .	191
5.14.2.2 DECODE_PORT . . . . .	191
5.14.2.3 ENCODE_PORT . . . . .	191
5.14.2.4 PORT_CNF_IN . . . . .	192
5.14.2.5 PORT_CNF_OUT . . . . .	192
5.14.3 Variable Documentation . . . . .	192
5.14.3.1 p_global_BSW_dio . . . . .	192
5.15 dio_port_cnf.h File Reference . . . . .	192
5.15.1 Detailed Description . . . . .	193
5.15.2 Macro Definition Documentation . . . . .	193
5.15.2.1 PORT_A . . . . .	193
5.15.2.2 PORT_B . . . . .	193
5.15.2.3 PORT_C . . . . .	194
5.15.2.4 PORT_D . . . . .	194

5.15.2.5 PORTB_CNF_DDRB . . . . .	194
5.15.2.6 PORTB_CNF_PORTB . . . . .	194
5.16 dio_reg_atm2560.h File Reference . . . . .	195
5.16.1 Detailed Description . . . . .	195
5.16.2 Macro Definition Documentation . . . . .	195
5.16.2.1 DDRA_PTR . . . . .	196
5.16.2.2 DDRB_PTR . . . . .	196
5.16.2.3 DDRC_PTR . . . . .	196
5.16.2.4 DDRD_PTR . . . . .	196
5.16.2.5 PINA_PTR . . . . .	196
5.16.2.6 PINB_PTR . . . . .	197
5.16.2.7 PINC_PTR . . . . .	197
5.16.2.8 PIND_PTR . . . . .	197
5.16.2.9 PORTA_PTR . . . . .	197
5.16.2.10 PORTB_PTR . . . . .	197
5.16.2.11 PORTC_PTR . . . . .	198
5.16.2.12 PORTD_PTR . . . . .	198
5.17 DisplayInterface.cpp File Reference . . . . .	198
5.17.1 Detailed Description . . . . .	198
5.17.2 Variable Documentation . . . . .	199
5.17.2.1 p_global_ASW_DisplayInterface . . . . .	199
5.18 DisplayInterface.h File Reference . . . . .	199
5.18.1 Detailed Description . . . . .	200
5.18.2 Macro Definition Documentation . . . . .	200
5.18.2.1 DISPLAY_LINE_SHIFT_PERIOD_MS . . . . .	200
5.18.2.2 DISPLAY_LINE_SHIFT_TEMPO_TIME . . . . .	200
5.18.3 Enumeration Type Documentation . . . . .	200
5.18.3.1 T_DisplayInterface_LineAlignment . . . . .	200
5.18.3.2 T_DisplayInterface_LineDisplayMode . . . . .	201
5.18.4 Variable Documentation . . . . .	201

---

5.18.4.1 p_global_ASW_DisplayInterface . . . . .	201
5.19 DisplayManagement.cpp File Reference . . . . .	202
5.19.1 Detailed Description . . . . .	202
5.19.2 Variable Documentation . . . . .	202
5.19.2.1 noSensorsDisplayString . . . . .	202
5.19.2.2 p_global_ASW_DisplayManagement . . . . .	203
5.19.2.3 welcomeMessageString . . . . .	203
5.20 DisplayManagement.h File Reference . . . . .	203
5.20.1 Detailed Description . . . . .	204
5.20.2 Macro Definition Documentation . . . . .	204
5.20.2.1 DISPLAY_MGT_FIRST_LINE_SENSORS . . . . .	204
5.20.2.2 DISPLAY_MGT_I2C_BITRATE . . . . .	204
5.20.2.3 DISPLAY_MGT_LCD_I2C_ADDR . . . . .	204
5.20.2.4 DISPLAY_MGT_PERIOD_TASK_SENSOR . . . . .	205
5.20.2.5 DISPLAY_MGT_PERIOD_WELCOME_MSG_REMOVAL . . . . .	205
5.20.3 Variable Documentation . . . . .	205
5.20.3.1 LCD_init_cnf . . . . .	205
5.20.3.2 p_global_ASW_DisplayManagement . . . . .	205
5.21 HumSensor.cpp File Reference . . . . .	206
5.21.1 Detailed Description . . . . .	206
5.21.2 Macro Definition Documentation . . . . .	206
5.21.2.1 DHT22_PORT . . . . .	206
5.22 HumSensor.h File Reference . . . . .	207
5.22.1 Detailed Description . . . . .	207
5.23 I2C.cpp File Reference . . . . .	207
5.23.1 Detailed Description . . . . .	208
5.23.2 Variable Documentation . . . . .	208
5.23.2.1 p_global_BSW_i2c . . . . .	208
5.24 I2C.h File Reference . . . . .	208
5.24.1 Detailed Description . . . . .	209

---

---

5.24.2 Macro Definition Documentation . . . . .	209
5.24.2.1 DATA_ACK . . . . .	209
5.24.2.2 SLA_ACK . . . . .	209
5.24.2.3 START . . . . .	210
5.24.3 Variable Documentation . . . . .	210
5.24.3.1 p_global_BSW_i2c . . . . .	210
5.25 int.cpp File Reference . . . . .	210
5.25.1 Detailed Description . . . . .	211
5.25.2 Function Documentation . . . . .	211
5.25.2.1 ISR() [1/2] . . . . .	211
5.25.2.2 ISR() [2/2] . . . . .	212
5.26 keepAliveLed.cpp File Reference . . . . .	212
5.26.1 Detailed Description . . . . .	213
5.26.2 Variable Documentation . . . . .	213
5.26.2.1 p_global_ASW_keepAliveLed . . . . .	213
5.27 keepAliveLed.h File Reference . . . . .	213
5.27.1 Detailed Description . . . . .	214
5.27.2 Macro Definition Documentation . . . . .	214
5.27.2.1 LED_PORT . . . . .	214
5.27.2.2 PERIOD_MS_TASK_LED . . . . .	214
5.27.3 Variable Documentation . . . . .	215
5.27.3.1 p_global_ASW_keepAliveLed . . . . .	215
5.28 LCD.cpp File Reference . . . . .	215
5.28.1 Detailed Description . . . . .	215
5.28.2 Variable Documentation . . . . .	216
5.28.2.1 p_global_BSW_lcd . . . . .	216
5.29 LCD.h File Reference . . . . .	216
5.29.1 Detailed Description . . . . .	218
5.29.2 Macro Definition Documentation . . . . .	218
5.29.2.1 BACKLIGHT_PIN . . . . .	218

---

5.29.2.2 EN_PIN . . . . .	218
5.29.2.3 LCD_CNF_BACKLIGHT_OFF . . . . .	218
5.29.2.4 LCD_CNF_BACKLIGHT_ON . . . . .	219
5.29.2.5 LCD_CNF_CURSOR_BLINK_OFF . . . . .	219
5.29.2.6 LCD_CNF_CURSOR_BLINK_ON . . . . .	219
5.29.2.7 LCD_CNF_CURSOR_OFF . . . . .	219
5.29.2.8 LCD_CNF_CURSOR_ON . . . . .	219
5.29.2.9 LCD_CNF_DISPLAY_OFF . . . . .	220
5.29.2.10 LCD_CNF_DISPLAY_ON . . . . .	220
5.29.2.11 LCD_CNF_ENTRY_MODE_DIRECTION_LEFT . . . . .	220
5.29.2.12 LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT . . . . .	220
5.29.2.13 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF . . . . .	220
5.29.2.14 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON . . . . .	221
5.29.2.15 LCD_CNF_FONT_5_11 . . . . .	221
5.29.2.16 LCD_CNF_FONT_5_8 . . . . .	221
5.29.2.17 LCD_CNF_ONE_LINE . . . . .	221
5.29.2.18 LCD_CNF_SHIFT_ID . . . . .	221
5.29.2.19 LCD_CNF_SHIFT_SH . . . . .	222
5.29.2.20 LCD_CNF_TWO_LINE . . . . .	222
5.29.2.21 LCD_DISPLAY_CTRL_FIELD_B . . . . .	222
5.29.2.22 LCD_DISPLAY_CTRL_FIELD_C . . . . .	222
5.29.2.23 LCD_DISPLAY_CTRL_FIELD_D . . . . .	222
5.29.2.24 LCD_FCT_SET_FIELD_DL . . . . .	223
5.29.2.25 LCD_FCT_SET_FIELD_F . . . . .	223
5.29.2.26 LCD_FCT_SET_FIELD_N . . . . .	223
5.29.2.27 LCD_INST_CLR_DISPLAY_BIT . . . . .	223
5.29.2.28 LCD_INST_DISPLAY_CTRL . . . . .	223
5.29.2.29 LCD_INST_ENTRY_MODE_SET . . . . .	224
5.29.2.30 LCD_INST_FUNCTION_SET . . . . .	224
5.29.2.31 LCD_INST_SET_DDRAM_ADDR . . . . .	224

---

5.29.2.32 LCD_RAM_1_LINE_MAX . . . . .	224
5.29.2.33 LCD_RAM_1_LINE_MIN . . . . .	224
5.29.2.34 LCD_RAM_2_LINES_MAX_1 . . . . .	225
5.29.2.35 LCD_RAM_2_LINES_MAX_2 . . . . .	225
5.29.2.36 LCD_RAM_2_LINES_MIN_1 . . . . .	225
5.29.2.37 LCD_RAM_2_LINES_MIN_2 . . . . .	225
5.29.2.38 LCD_SIZE_NB_CHAR_PER_LINE . . . . .	225
5.29.2.39 LCD_SIZE_NB_LINES . . . . .	226
5.29.2.40 LCD_WAIT_CLR_RETURN . . . . .	226
5.29.2.41 LCD_WAIT_OTHER_MODES . . . . .	226
5.29.2.42 RS_PIN . . . . .	226
5.29.2.43 RW_PIN . . . . .	226
5.29.3 Enumeration Type Documentation . . . . .	226
5.29.3.1 T_LCD_command . . . . .	226
5.29.3.2 T_LCD_config_mode . . . . .	227
5.29.3.3 T_LCD_ram_area . . . . .	227
5.29.4 Variable Documentation . . . . .	227
5.29.4.1 p_global_BSW_lcd . . . . .	228
5.30 LinkedList.cpp File Reference . . . . .	228
5.30.1 Detailed Description . . . . .	228
5.31 LinkedList.h File Reference . . . . .	228
5.31.1 Detailed Description . . . . .	229
5.31.2 Typedef Documentation . . . . .	229
5.31.2.1 CompareFctPtr_t . . . . .	229
5.32 main.cpp File Reference . . . . .	229
5.32.1 Detailed Description . . . . .	230
5.32.2 Macro Definition Documentation . . . . .	230
5.32.2.1 DEBUG_ACTIVE_PORT . . . . .	230
5.32.3 Function Documentation . . . . .	230
5.32.3.1 main() . . . . .	231

---

5.32.4 Variable Documentation . . . . .	231
5.32.4.1 ASW_init_cnf . . . . .	231
5.32.4.2 isDebugModeActivated . . . . .	231
5.33 main.h File Reference . . . . .	232
5.33.1 Detailed Description . . . . .	232
5.33.2 Variable Documentation . . . . .	232
5.33.2.1 ASW_init_cnf . . . . .	232
5.33.2.2 isDebugModeActivated . . . . .	233
5.34 operators.cpp File Reference . . . . .	233
5.34.1 Detailed Description . . . . .	233
5.34.2 Function Documentation . . . . .	234
5.34.2.1 operator delete() . . . . .	234
5.34.2.2 operator new() . . . . .	234
5.35 operators.h File Reference . . . . .	234
5.35.1 Detailed Description . . . . .	235
5.35.2 Function Documentation . . . . .	235
5.35.2.1 operator delete() . . . . .	235
5.35.2.2 operator new() . . . . .	236
5.36 scheduler.cpp File Reference . . . . .	236
5.36.1 Detailed Description . . . . .	237
5.36.2 Variable Documentation . . . . .	237
5.36.2.1 p_global_scheduler . . . . .	237
5.37 scheduler.h File Reference . . . . .	237
5.37.1 Detailed Description . . . . .	238
5.37.2 Macro Definition Documentation . . . . .	238
5.37.2.1 PRESCALER_PERIODIC_TIMER . . . . .	238
5.37.2.2 SW_PERIOD_MS . . . . .	238
5.37.2.3 TIMER_CTC_VALUE . . . . .	239
5.37.3 Typedef Documentation . . . . .	239
5.37.3.1 TaskPtr_t . . . . .	239

5.37.4 Variable Documentation . . . . .	239
5.37.4.1 p_global_scheduler . . . . .	239
5.38 Sensor.cpp File Reference . . . . .	239
5.38.1 Detailed Description . . . . .	240
5.38.2 Macro Definition Documentation . . . . .	240
5.38.2.1 TASK_PERIOD_DEFAULT . . . . .	240
5.38.2.2 VALIDITY_TIMEOUT_MS_DEFAULT . . . . .	240
5.39 Sensor.h File Reference . . . . .	240
5.39.1 Detailed Description . . . . .	241
5.40 sensor_configuration.cpp File Reference . . . . .	241
5.40.1 Detailed Description . . . . .	242
5.40.2 Macro Definition Documentation . . . . .	242
5.40.2.1 SENSOR_MGT_CNF_DEFAULT_PERIOD . . . . .	242
5.40.2.2 SENSOR_MGT_CNF_DEFAULT_TMO . . . . .	242
5.40.3 Variable Documentation . . . . .	242
5.40.3.1 SensorManagement_Sensor_Config_list . . . . .	243
5.41 sensor_configuration.h File Reference . . . . .	243
5.41.1 Detailed Description . . . . .	244
5.41.2 Variable Documentation . . . . .	244
5.41.2.1 SensorManagement_Sensor_Config_list . . . . .	244
5.42 SensorManagement.cpp File Reference . . . . .	244
5.42.1 Detailed Description . . . . .	245
5.42.2 Variable Documentation . . . . .	245
5.42.2.1 p_global_ASW_SensorManagement . . . . .	245
5.43 SensorManagement.h File Reference . . . . .	245
5.43.1 Detailed Description . . . . .	246
5.43.2 Enumeration Type Documentation . . . . .	246
5.43.2.1 T_SensorManagement_Sensor_Type . . . . .	246
5.43.3 Variable Documentation . . . . .	246
5.43.3.1 p_global_ASW_SensorManagement . . . . .	246

5.44 String.cpp File Reference . . . . .	247
5.44.1 Detailed Description . . . . .	247
5.45 String.h File Reference . . . . .	247
5.45.1 Detailed Description . . . . .	248
5.46 TempSensor.cpp File Reference . . . . .	248
5.46.1 Detailed Description . . . . .	248
5.46.2 Macro Definition Documentation . . . . .	249
5.46.2.1 DHT22_PORT . . . . .	249
5.47 TempSensor.h File Reference . . . . .	249
5.47.1 Detailed Description . . . . .	249
5.48 timer.cpp File Reference . . . . .	250
5.48.1 Detailed Description . . . . .	250
5.48.2 Variable Documentation . . . . .	250
5.48.2.1 p_global_BSW_timer . . . . .	250
5.49 timer.h File Reference . . . . .	251
5.49.1 Detailed Description . . . . .	251
5.49.2 Variable Documentation . . . . .	251
5.49.2.1 p_global_BSW_timer . . . . .	251
5.50 usart.cpp File Reference . . . . .	252
5.50.1 Detailed Description . . . . .	252
5.50.2 Variable Documentation . . . . .	252
5.50.2.1 p_global_BSW_usart . . . . .	252
5.51 usart.h File Reference . . . . .	253
5.51.1 Detailed Description . . . . .	253
5.51.2 Variable Documentation . . . . .	253
5.51.2.1 p_global_BSW_usart . . . . .	253
5.52 Watchdog.cpp File Reference . . . . .	254
5.52.1 Detailed Description . . . . .	254
5.52.2 Macro Definition Documentation . . . . .	254
5.52.2.1 WDG_TIMEOUT_DEFAULT_MS . . . . .	255

---

5.52.3 Variable Documentation . . . . .	255
5.52.3.1 p_global_BSW_wdg . . . . .	255
5.53 Watchdog.h File Reference . . . . .	255
5.53.1 Detailed Description . . . . .	256
5.53.2 Macro Definition Documentation . . . . .	256
5.53.2.1 WDG_TMO_120MS . . . . .	256
5.53.2.2 WDG_TMO_15MS . . . . .	257
5.53.2.3 WDG_TMO_1S . . . . .	257
5.53.2.4 WDG_TMO_250MS . . . . .	257
5.53.2.5 WDG_TMO_2S . . . . .	257
5.53.2.6 WDG_TMO_30MS . . . . .	257
5.53.2.7 WDG_TMO_4S . . . . .	258
5.53.2.8 WDG_TMO_500MS . . . . .	258
5.53.2.9 WDG_TMO_60MS . . . . .	258
5.53.2.10 WDG_TMO_8S . . . . .	258
5.53.3 Variable Documentation . . . . .	258
5.53.3.1 p_global_BSW_wdg . . . . .	258
Index . . . . .	259



# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

CpuLoad . . . . .	9
debug_mgt_state_struct_t . . . . .	14
DebugInterface . . . . .	15
DebugManagement . . . . .	22
dht22 . . . . .	34
dio . . . . .	41
DisplayInterface . . . . .	50
DisplayManagement . . . . .	63
I2C . . . . .	72
keepAliveLed . . . . .	76
LCD . . . . .	78
LinkedList . . . . .	95
scheduler . . . . .	102
Sensor . . . . .	112
HumSensor . . . . .	68
TempSensor . . . . .	146
SensorManagement . . . . .	120
String . . . . .	125
T_ASW_init_cnf . . . . .	135
T_display_data . . . . .	136
T_Display_shift_data . . . . .	138
T_LCD_conf_struct . . . . .	140
LinkedList::T_LL_element . . . . .	142
T_SensorManagement_Sensor_Config . . . . .	143
scheduler::Task_t . . . . .	145
timer . . . . .	150
uart . . . . .	154
Watchdog . . . . .	159



# Chapter 2

## Class Index

### 2.1 Class List

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

<a href="#">CpuLoad</a>	Class defining CPU load libraries . . . . .	9
<a href="#">debug_mgt_state_struct_t</a>	Structure containing all debug states . . . . .	14
<a href="#">DebugInterface</a>	Class used for debugging on usart link . . . . .	15
<a href="#">DebugManagement</a>	Debug management class . . . . .	22
<a href="#">dht22</a>	DHT 22 driver class . . . . .	34
<a href="#">dio</a>	DIO class . . . . .	41
<a href="#">DisplayInterface</a>	Display interface services class . . . . .	50
<a href="#">DisplayManagement</a>	Display management class . . . . .	63
<a href="#">HumSensor</a>	Class for humidity sensor . . . . .	68
<a href="#">I2C</a>	Two-wire serial interface ( <a href="#">I2C</a> ) class definition . . . . .	72
<a href="#">keepAliveLed</a>	Class for keep-alive LED blinking . . . . .	76
<a href="#">LCD</a>	Class for <a href="#">LCD</a> S2004A display driver . . . . .	78
<a href="#">LinkedList</a>	Linked list class . . . . .	95
<a href="#">scheduler</a>	Scheduler class . . . . .	102
<a href="#">Sensor</a>	Generic class for sensor device . . . . .	112
<a href="#">SensorManagement</a>	<a href="#">Sensor</a> management class . . . . .	120
<a href="#">String</a>	<a href="#">String</a> management class . . . . .	125
<a href="#">T_ASW_init_cnf</a>	ASW initialization configuration structure . . . . .	135

<a href="#">T_display_data</a>	Structure containing display data . . . . .	136
<a href="#">T_Display_shift_data</a>	Structure containing shift data . . . . .	138
<a href="#">T_LCD_conf_struct</a>	Structure defining <a href="#">LCD</a> configuration . . . . .	140
<a href="#">LinkedList::T_LL_element</a>	Type defining a linked list element . . . . .	142
<a href="#">T_SensorManagement_Sensor_Config</a>	Sensor informations structure . . . . .	143
<a href="#">scheduler::Task_t</a>	Type defining a task structure . . . . .	145
<a href="#">TempSensor</a>	Class for temperature sensor . . . . .	146
<a href="#">timer</a>	Class defining a timer . . . . .	150
<a href="#">usart</a>	USART serial bus class . . . . .	154
<a href="#">Watchdog</a>	<a href="#">Watchdog</a> management class . . . . .	159

# Chapter 3

## File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

<code>asw.cpp</code>	ASW main file . . . . .	169
<code>asw.h</code>	ASW main header file . . . . .	171
<code>bsw.cpp</code>	BSW main file . . . . .	172
<code>bsw.h</code>	BSW main header file . . . . .	174
<code>CpuLoad.cpp</code>	Defines functions of class <code>CpuLoad</code> . . . . .	175
<code>CpuLoad.h</code>	<code>CpuLoad</code> class header file . . . . .	176
<code>DebugInterface.cpp</code>	This file defines classes for log and debug data transmission on USART link . . . . .	178
<code>DebugInterface.h</code>	Header file for debug and logging functions . . . . .	179
<code>DebugManagement.cpp</code>	Debug management class source file . . . . .	180
<code>DebugManagement.h</code>	Debug management class header file . . . . .	184
<code>dht22.cpp</code>	This file defines classes for DHT22 driver . . . . .	186
<code>dht22.h</code>	DHT22 driver header file . . . . .	188
<code>dio.cpp</code>	DIO library . . . . .	189
<code>dio.h</code>	DIO library header file . . . . .	190
<code>dio_port_cnf.h</code>	Digital ports configuration file . . . . .	192
<code>dio_reg_atm2560.h</code>	Defines DIO register addresses for ATMEGA2560 . . . . .	195
<code>DisplayInterface.cpp</code>	Source code file for display services . . . . .	198
<code>DisplayInterface.h</code>	<code>DisplayInterface</code> class header file . . . . .	199

DisplayManagement.cpp	Display management source file	202
DisplayManagement.h	Display management class header file	203
HumSensor.cpp	Defines function of class <code>HumSensor</code>	206
HumSensor.h	Class <code>HumSensor</code> header file	207
I2C.cpp	Two-wire interface ( <code>I2C</code> ) source file	207
I2C.h	<code>I2C</code> class header file	208
int.cpp	Interrupt management source file	210
keepAliveLed.cpp	Definition of function for class <code>keepAliveLed</code>	212
keepAliveLed.h	Class <code>keepAliveLed</code> header file	213
LCD.cpp	<code>LCD</code> class source file	215
LCD.h	<code>LCD</code> class header file	216
LinkedList.cpp	Linked List library source file	228
LinkedList.h	Linked List library header file	228
main.cpp	Background task file	229
main.h	Background task header file	232
operators.cpp	C++ operators definitions	233
operators.h	C++ operators definitions header file	234
scheduler.cpp	Defines scheduler class	236
scheduler.h	Scheduler class header file	237
Sensor.cpp	<code>Sensor</code> class source code file	239
Sensor.h	<code>Sensor</code> class header file	240
sensor_configuration.cpp	<code>Sensor</code> configuration file	241
sensor_configuration.h	Sensors configuration header file	243
SensorManagement.cpp	<code>SensorManagement</code> class source code file	244
SensorManagement.h	<code>SensorManagement</code> class header file	245
String.cpp	<code>String</code> class source file	247
String.h	<code>String</code> class header file	247
TempSensor.cpp	Defines function of class <code>TempSensor</code>	248
TempSensor.h	Class <code>TempSensor</code> header file	249

<a href="#">timer.cpp</a>	Defines function for class timer . . . . .	250
<a href="#">timer.h</a>	Timer class header file . . . . .	251
<a href="#">usart.cpp</a>	BSW library for USART . . . . .	252
<a href="#">usart.h</a>	Header file for USART library . . . . .	253
<a href="#">Watchdog.cpp</a>	Class <a href="#">Watchdog</a> source code file . . . . .	254
<a href="#">Watchdog.h</a>	Class <a href="#">Watchdog</a> header file . . . . .	255



# Chapter 4

## Class Documentation

### 4.1 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](#)

#### 4.1.1 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.

## 4.1.2 Constructor & Destructor Documentation

### 4.1.2.1 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.

## 4.1.3 Member Function Documentation

### 4.1.3.1 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

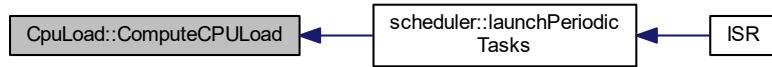
Nothing

Definition at line 40 of file CpuLoad.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.3.2 getAverageCPULoad()

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

Get average CPU load value.

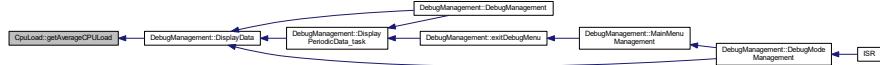
This function returns the average CPU load value

##### Returns

Average CPU load value

Definition at line 58 of file CpuLoad.h.

Here is the caller graph for this function:



#### 4.1.3.3 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:



#### 4.1.3.4 getMaxCPUload()

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

Get maximum CPU load value.

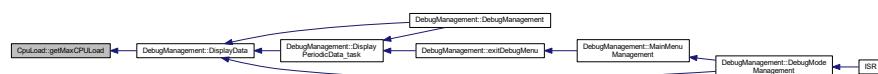
This function returns the maximum CPU load value

##### Returns

Maximum CPU load value

Definition at line 69 of file CpuLoad.h.

Here is the caller graph for this function:



#### 4.1.4 Member Data Documentation

##### 4.1.4.1 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.

##### 4.1.4.2 current\_load

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

Current CPU load (load of last cycle)

Definition at line 75 of file CpuLoad.h.

#### 4.1.4.3 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.

#### 4.1.4.4 max\_load

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

Maximum CPU load since power on

Definition at line 77 of file CpuLoad.h.

#### 4.1.4.5 sample\_cnt

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

Number of samples used to compute average load

Definition at line 78 of file CpuLoad.h.

#### 4.1.4.6 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.

#### 4.1.4.7 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](#)

## 4.2 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`

#### 4.2.1 Detailed Description

Structure containing all debug states.

Definition at line 40 of file DebugManagement.h.

#### 4.2.2 Member Data Documentation

##### 4.2.2.1 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.

##### 4.2.2.2 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:

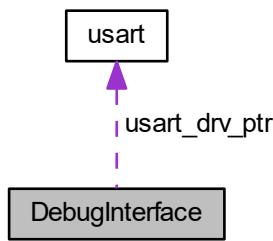
- [DebugManagement.h](#)

## 4.3 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](#)

### 4.3.1 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.

### 4.3.2 Constructor & Destructor Documentation

#### 4.3.2.1 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.

### 4.3.3 Member Function Documentation

#### 4.3.3.1 ClearScreen()

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

Screen clearing function.

This function clears the entire display by sending the \f 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:



#### 4.3.3.2 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 `\n` and `\r` 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:



#### 4.3.3.3 read()

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

USART read function.

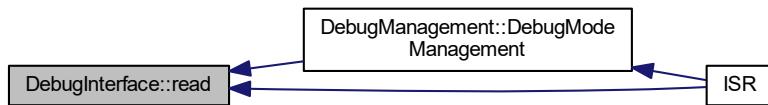
This function will read the last received byte on USART link

##### Returns

Received byte

Definition at line 82 of file DebugInterface.h.

Here is the caller graph for this function:



#### 4.3.3.4 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

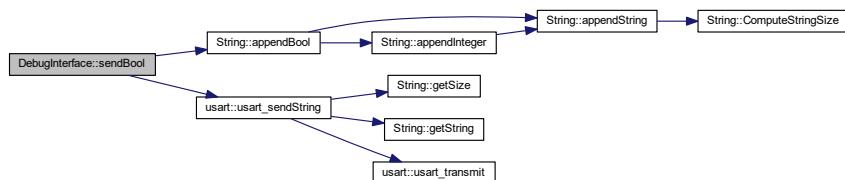
in	<code>data</code>	boolean data to be sent
in	<code>isText</code>	<a href="#">String</a> conversion configuration

##### Returns

Nothing

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

Here is the call graph for this function:



#### 4.3.3.5 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**

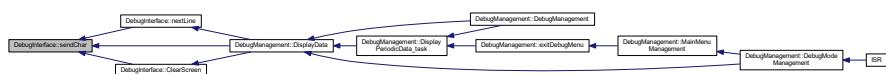
Nothing

Definition at line 44 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.3.3.6 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**

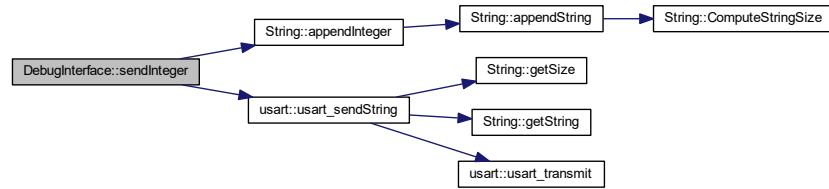
<b>in</b>	<b>data</b>	integer data to be sent
<b>in</b>	<b>base</b>	numerical base used to convert integer into string (between 2 and 36)

**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:



#### 4.3.3.7 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

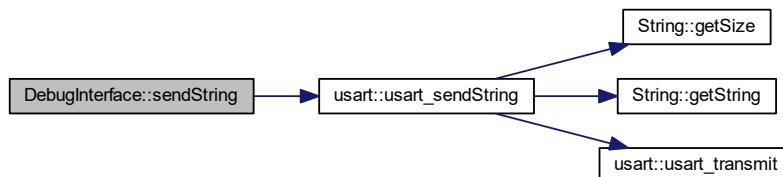
in	<i>str</i>	Pointer to the string being sent
----	------------	----------------------------------

##### Returns

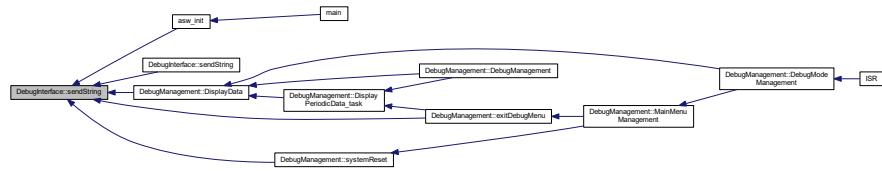
Nothing

Definition at line 31 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.3.8 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

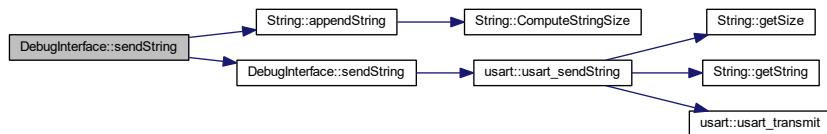
in	<i>str</i>	Pointer to the chain to send.
----	------------	-------------------------------

##### Returns

Nothing

Definition at line 37 of file DebugInterface.cpp.

Here is the call graph for this function:



#### 4.3.4 Member Data Documentation

#### 4.3.4.1 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:

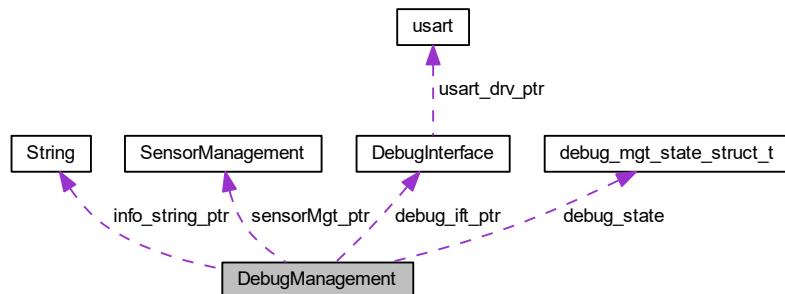
- [DebugInterface.h](#)
- [DebugInterface.cpp](#)

## 4.4 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.*
- [String \\* getInfoStringPtr \(\)](#)  
*Info string get function.*
- [void setInfoStringPtr \(String \\*addr\)](#)  
*Info message setting function.*

## Static Public Member Functions

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

*Displays periodic data on usart link.*

## Private Member Functions

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

## Private Attributes

- [DebugInterface \\* debug\\_ift\\_ptr](#)
- [SensorManagement \\* sensorMgt\\_ptr](#)
- [uint8\\_t \\* menu\\_string\\_ptr](#)
- [String \\* info\\_string\\_ptr](#)
- [debug\\_mgt\\_state\\_struct\\_t debug\\_state](#)
- bool [isInfoStringDisplayed](#)

### 4.4.1 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.

### 4.4.2 Constructor & Destructor Documentation

#### 4.4.2.1 DebugManagement()

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

Class constructor.

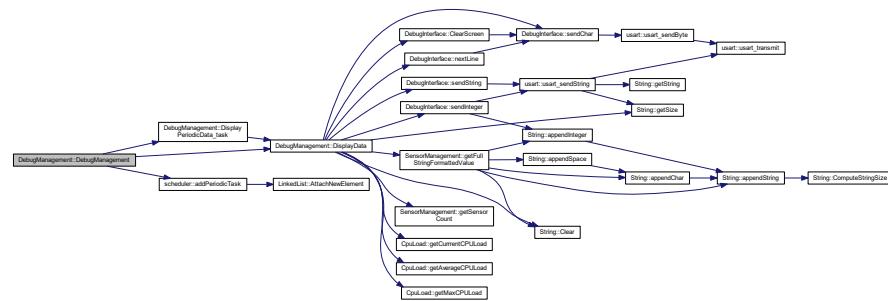
This function initializes the class. If needed, it creates a new instance of debug interface object.

##### Returns

Nothing

Definition at line 103 of file DebugManagement.cpp.

Here is the call graph for this function:



#### 4.4.3 Member Function Documentation

##### 4.4.3.1 DebugModeManagement()

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

Management of debug mode.

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

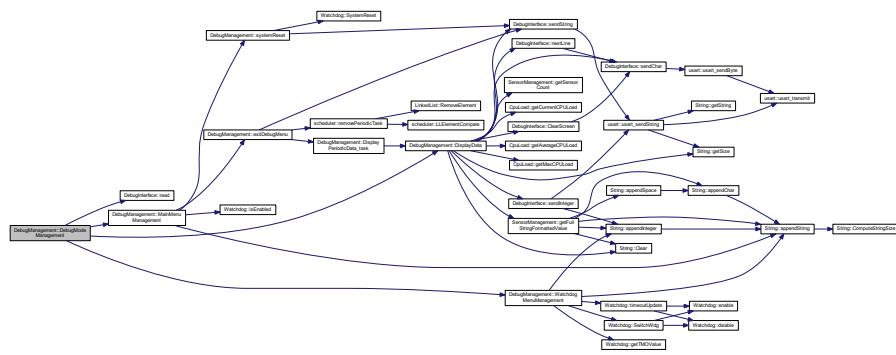
- MAIN\_MENU state : handles user choice in main menu and selects next state
- WDG\_MENU state : handles user choice in watchdog menu and selects next state  
It is called each time a data is received on USART and debug mode is active.

## Returns

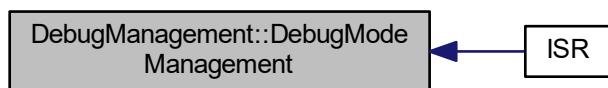
True if the debug mode shall be closed, false otherwise

Definition at line 203 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.4.3.2 DisplayData()

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

Displays data on usart link.

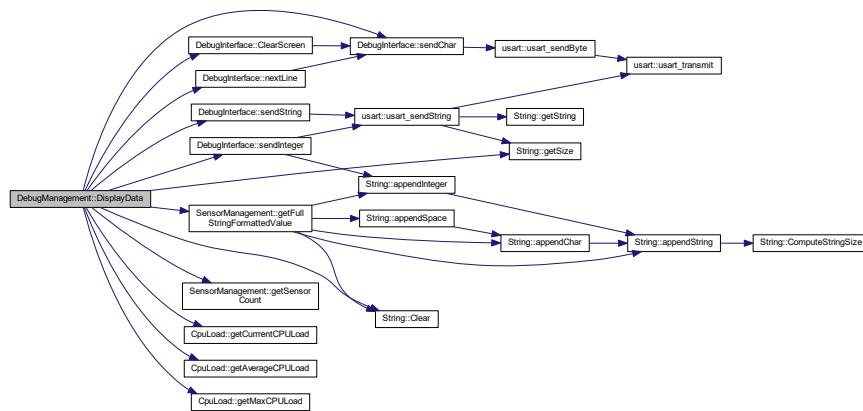
This task displays the menu and periodic data (temperature, humidity and CPU load) on usart screen.

**Returns**

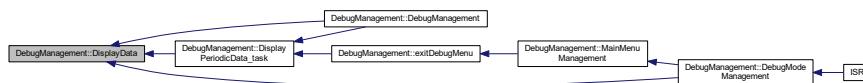
Nothing

Definition at line 132 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.4.3.3 DisplayPeriodicData\_task()

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

Displays periodic data on usart link.

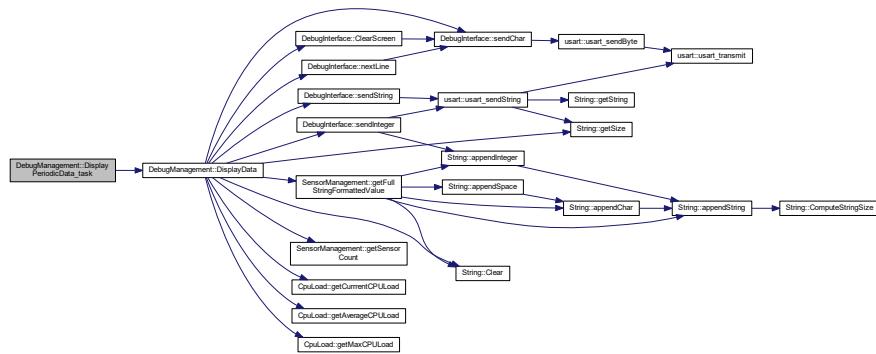
This task displays the menu and periodic data (temperature, humidity and CPU load) on usart screen. It only calls the function `DisplayData`.

**Returns**

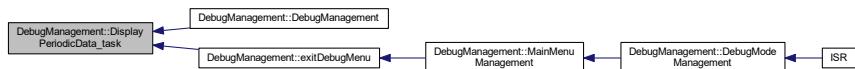
Nothing

Definition at line 195 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.4.3.4 exitDebugMenu()**

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

Debug menu exit function.

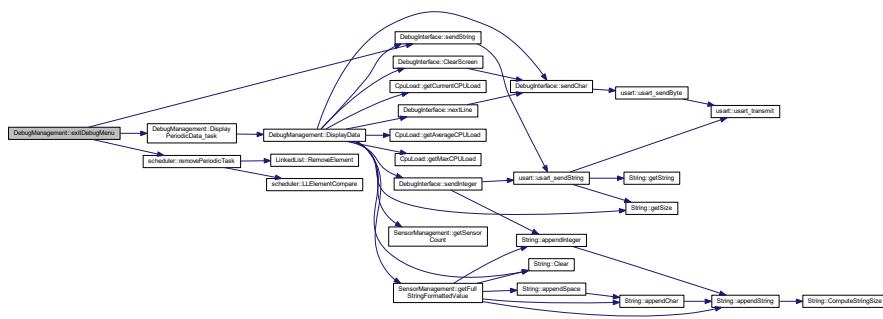
This function prepares the exit of debug menu. It writes the message "Bye !" on the screen and removes the periodic task from the scheduler.

**Returns**

Nothing.

Definition at line 230 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.4.3.5 getIfpt()

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

Interface pointer get function.

This function returns the pointer to the debug interface object

##### Returns

Pointer to debug interface

Definition at line 95 of file DebugManagement.h.

#### 4.4.3.6 getInfoStringPtr()

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

Info string get function.

This function returns the pointer to the info string to display

##### Returns

Info string pointer

Definition at line 115 of file DebugManagement.h.

#### 4.4.3.7 getMenuStringPtr()

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

Menu string get function.

This function returns the pointer to the menu string to display

##### Returns

Menu string pointer

Definition at line 105 of file DebugManagement.h.

#### 4.4.3.8 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

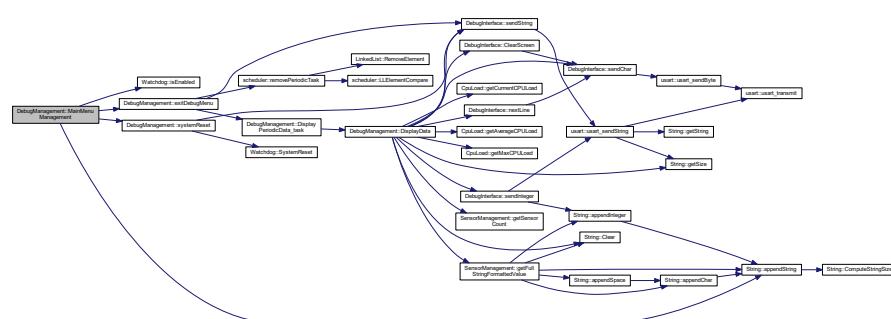
in	<i>rcv_char</i>	Character received on USART bus.
----	-----------------	----------------------------------

##### Returns

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

Definition at line 347 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.4.3.9 setInfoStringPtr()

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

Info message setting function.

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

##### Parameters

in	addr	String address
----	------	----------------

##### Returns

Nothing

Definition at line 126 of file DebugManagement.h.

#### 4.4.3.10 systemReset()

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

System reset function.

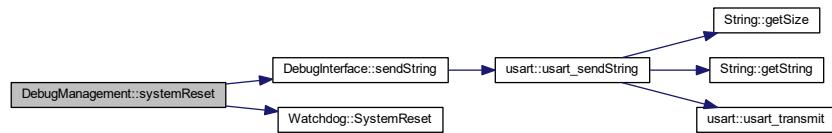
This function provokes a reset of the system. It displays a message on the screen and calls the reset function from watchdog class.

**Returns**

Nothing.

Definition at line 236 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.4.3.11 WatchdogMenuManagement()**

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

[Watchdog](#) menu management function.

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

**Parameters**

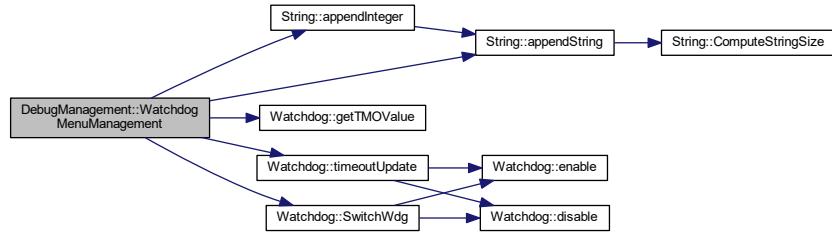
in	<code>rcv_char</code>	Character received on USART bus.
----	-----------------------	----------------------------------

**Returns**

Nothing.

Definition at line 242 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.4.4 Member Data Documentation

##### 4.4.4.1 debug\_ift\_ptr

`DebugInterface* DebugManagement::debug_ift_ptr [private]`

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

Definition at line 133 of file DebugManagement.h.

##### 4.4.4.2 debug\_state

`debug_mgt_state_struct_t DebugManagement::debug_state [private]`

Structure containing debug states for each menu

Definition at line 137 of file DebugManagement.h.

#### 4.4.4.3 info\_string\_ptr

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

Pointer to the info message to display

Definition at line 136 of file DebugManagement.h.

#### 4.4.4.4 isInfoStringDisplayed

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

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

Definition at line 138 of file DebugManagement.h.

#### 4.4.4.5 menu\_string\_ptr

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

Pointer to the current menu string to display

Definition at line 135 of file DebugManagement.h.

#### 4.4.4.6 sensorMgt\_ptr

```
SensorManagement* DebugManagement::sensorMgt_ptr [private]
```

Pointer to the sensor management object

Definition at line 134 of file DebugManagement.h.

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

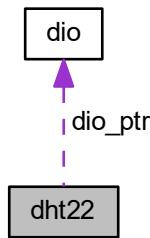
- [DebugManagement.h](#)
- [DebugManagement.cpp](#)

## 4.5 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 getTemperature (uint16_t *temperature)`  
*Temperature get function.*
- `bool getHumidity (uint16_t *humidity)`  
*Humidity get function.*

### Private Member Functions

- `void initializeCommunication ()`  
*Initializes the communication.*
- `void read ()`  
*Reads the data from DHT22.*

### Private Attributes

- `uint8_t dht22_port`
- `dio * dio_ptr`
- `uint16_t mem_temperature`
- `uint16_t mem_humidity`
- `bool mem_validity`
- `uint32_t pit_last_read`

### 4.5.1 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.

### 4.5.2 Constructor & Destructor Documentation

#### 4.5.2.1 dht22()

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

[dht22](#) class constructor.

Initializes the class [dht22](#).

#### Parameters

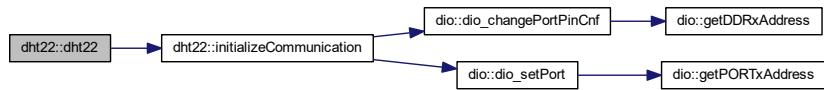
in	<i>port</i>	Encoded configuration of the port used for 1-wire communication.
----	-------------	--

#### Returns

Nothing

Definition at line 24 of file dht22.cpp.

Here is the call graph for this function:



### 4.5.3 Member Function Documentation

#### 4.5.3.1 getHumidity()

```
bool dht22::getHumidity (
    uint16_t * humidity )
```

Humidity get function.

This functions writes the humidity value at the given address and returns the validity of the data. If the values have not been refreshed during the current PIT, a read operation is performed on the DHT22 device.

## Parameters

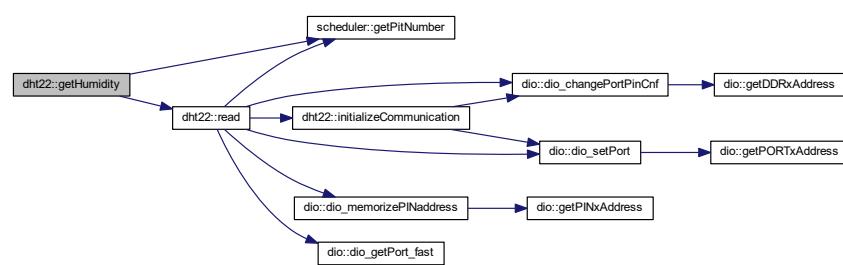
in	<i>humidity</i>	Address where the humidity shall be written
----	-----------------	---

## Returns

Validity of the data

Definition at line 220 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.5.3.2 getTemperature()

```
bool dht22::getTemperature (
    uint16_t * temperature )
```

Temperature get function.

This functions writes the temperature value at the given address and returns the validity of the data. If the values have not been refreshed during the current PIT, a read operation is performed on the DHT22 device.

## Parameters

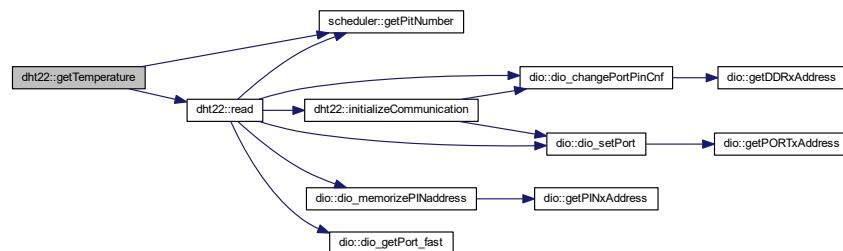
in	<i>temperature</i>	Address where the temperature shall be written
----	--------------------	--

**Returns**

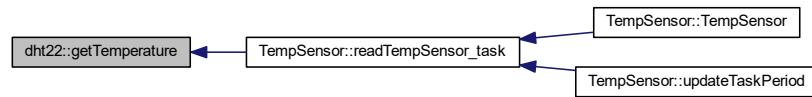
Validity of the data

Definition at line 231 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.5.3.3 initializeCommunication()

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

Initializes the communication.

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

**Returns**

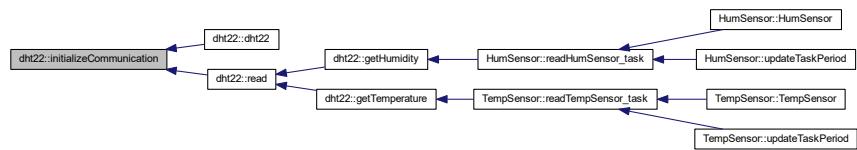
Nothing

Definition at line 210 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.5.3.4 read()

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

Reads the data from DHT22.

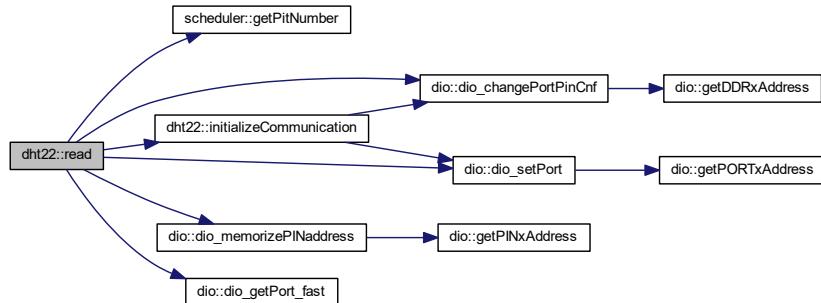
This function communicates with DHT22 using 1-wire protocol to read raw values of temperature and humidity. A checksum check is done when communication is finished to validate the received data. Validity of the data, temperature and humidity values are memorized in the associated class members.

## Returns

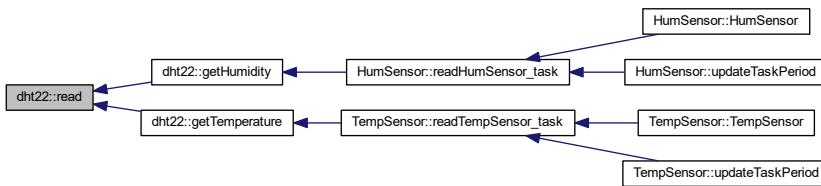
Nothing

Definition at line 34 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.5.4 Member Data Documentation

##### 4.5.4.1 dht22\_port

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

Variable containing the port used for 1-wire communication

Definition at line 55 of file dht22.h.

##### 4.5.4.2 dio\_ptr

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

Pointer to the DIO object

Definition at line 56 of file dht22.h.

##### 4.5.4.3 mem\_humidity

```
uint16_t dht22::mem_humidity [private]
```

Memorized value of humidity

Definition at line 58 of file dht22.h.

##### 4.5.4.4 mem\_temperature

```
uint16_t dht22::mem_temperature [private]
```

Memorized value of temperature

Definition at line 57 of file dht22.h.

##### 4.5.4.5 mem\_validity

```
bool dht22::mem_validity [private]
```

Memorized value of validity

Definition at line 59 of file dht22.h.

#### 4.5.4.6 pit\_last\_read

```
uint32_t dht22::pit_last_read [private]
```

Value of the PIT number when the last read operation has been performed

Definition at line 60 of file dht22.h.

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

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

## 4.6 dio Class Reference

DIO class.

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

### Public Member Functions

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

### Private Member Functions

- [\*\*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`

### 4.6.1 Detailed Description

DIO class.

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

Definition at line 27 of file dio.h.

### 4.6.2 Constructor & Destructor Documentation

#### 4.6.2.1 dio()

`dio::dio ()`

`dio` class constructor

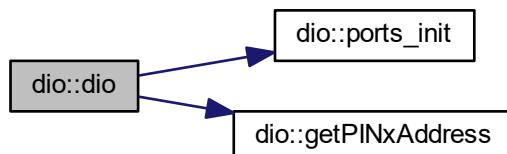
Initializes class `dio` and calls DIO hardware initialization function

Returns

Nothing

Definition at line 112 of file dio.cpp.

Here is the call graph for this function:



### 4.6.3 Member Function Documentation

#### 4.6.3.1 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**

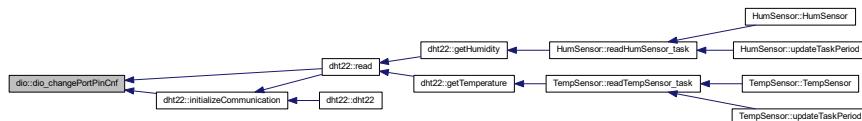
Nothing

Definition at line 149 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.6.3.2 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:

**4.6.3.3 dio\_getPort\_fast()**

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

Gets the logical state of the memorized pin.

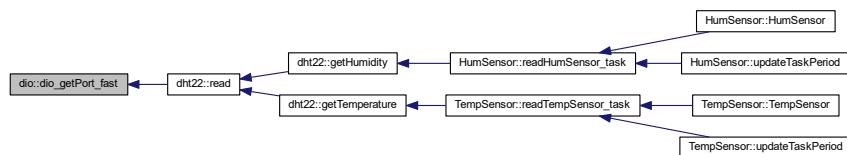
This function gets the logical value of the memorized pin. The corresponding port and pin index are stored in class members PINx\_addr\_mem and PINx\_idx\_mem. This mechanism is used to speed up reading time as this function no longer needs to extract register address and pin index from portcode.

**Returns**

Logical state of selected pin

Definition at line 171 of file dio.cpp.

Here is the caller graph for this function:



#### 4.6.3.4 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:



#### 4.6.3.5 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**

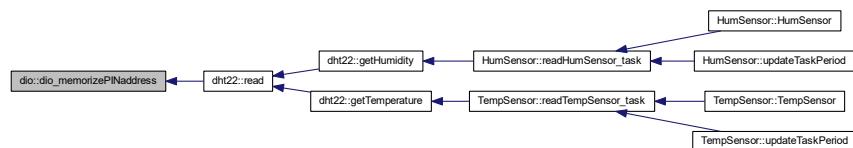
Nothing

Definition at line 165 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.6.3.6 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**

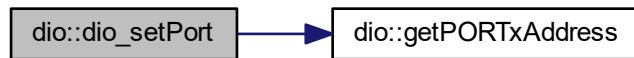
in	<i>portcode</i>	Encoded pin and register index
in	<i>state</i>	Requested state to set pin

**Returns**

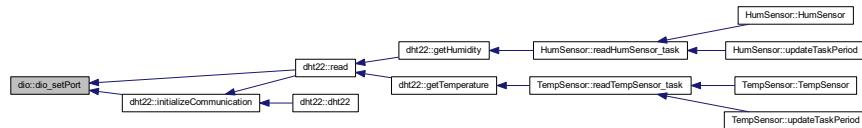
Nothing

Definition at line 121 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.6.3.7 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**

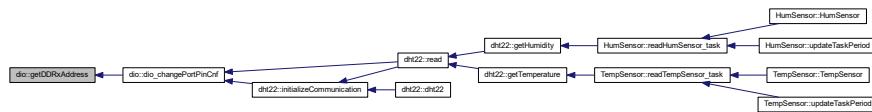
in	<i>portcode</i>	Encoded port code
----	-----------------	-------------------

**Returns**

Pointer to the DDRx register

Definition at line 83 of file dio.cpp.

Here is the caller graph for this function:



#### 4.6.3.8 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

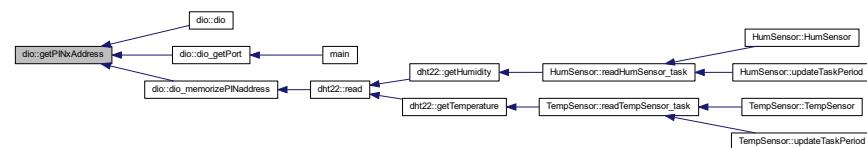
in	<i>portcode</i>	Encoded port code
----	-----------------	-------------------

##### Returns

Pointer to the PINx register

Definition at line 54 of file dio.cpp.

Here is the caller graph for this function:



#### 4.6.3.9 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

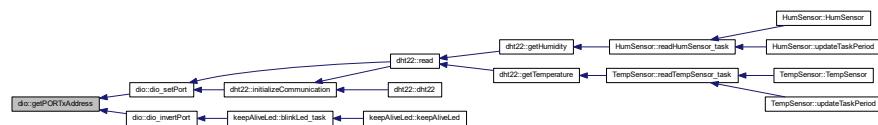
in	<i>portcode</i>	Encoded port code
----	-----------------	-------------------

## Returns

Pointer to the PORTx register

Definition at line 25 of file dio.cpp.

Here is the caller graph for this function:



## 4.6.3.10 ports\_init()

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

Digital ports hardware initialization function.

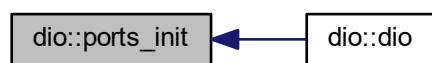
This function initializes digital ports as input or output and sets their initial values

## Returns

Nothing

Definition at line 16 of file dio.cpp.

Here is the caller graph for this function:



## 4.6.4 Member Data Documentation

#### 4.6.4.1 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.

#### 4.6.4.2 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:

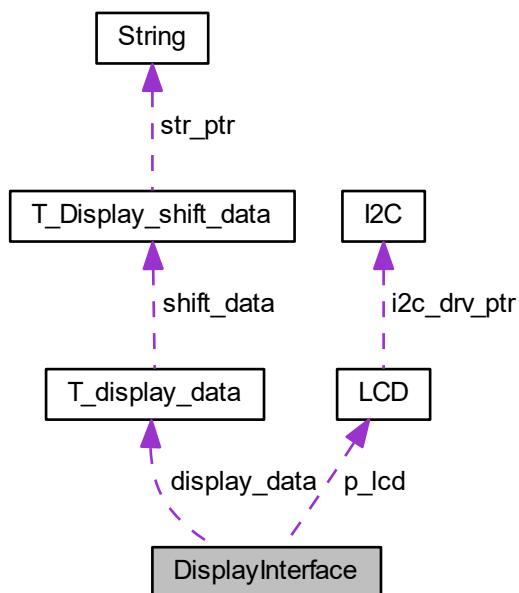
- [dio.h](#)
- [dio.cpp](#)

## 4.7 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 DisplayFullLine (String *str, 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.*
- `void updateLineAndRefresh (uint8_t *str, uint8_t size, uint8_t line)`  
*Line data string update 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.*

## Private Attributes

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

#### 4.7.1 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`.

#### 4.7.2 Constructor & Destructor Documentation

##### 4.7.2.1 `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

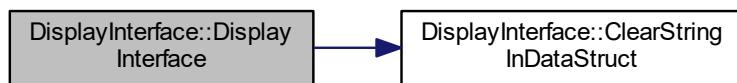
in	<i>LCD_init_cnf</i>	Initial configuration of the screen
----	---------------------	-------------------------------------

##### Returns

Nothing

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

Here is the call graph for this function:



#### 4.7.3 Member Function Documentation

#### 4.7.3.1 ClearFullScreen()

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

Screen cleaning function.

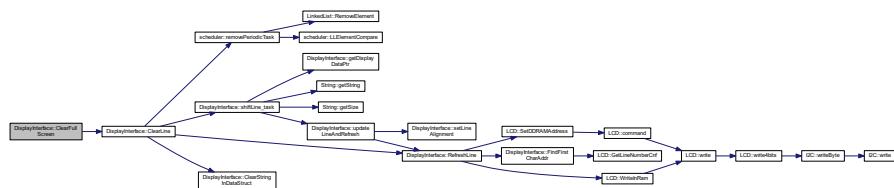
This function clears the entire display. It uses the ClearLine function on every line of the screen.

## Returns

Nothing

Definition at line 269 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.7.3.2 ClearLine()

```
    bool DisplayInterface::ClearLine (
```

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

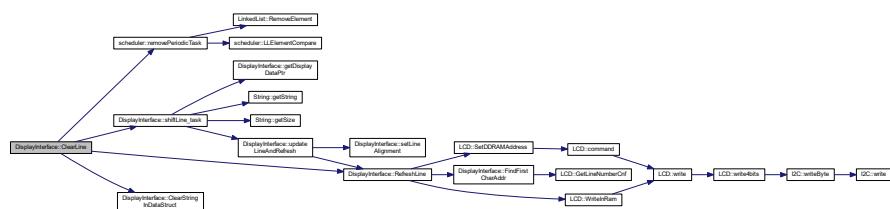
in	<i>line</i>	Line to clear
----	-------------	---------------

## Returns

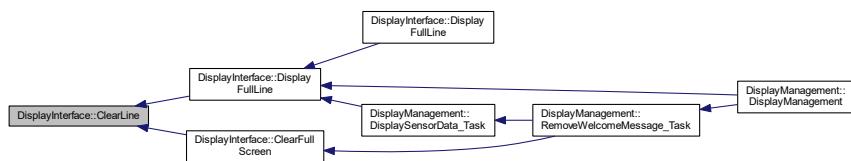
True if the line has been cleared, false otherwise

Definition at line 224 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.7.3.3 ClearStringInDataStruct()

```
void DisplayInterface::ClearStringInDataStruct (
```

## **String** data clearing structure.

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

## Parameters

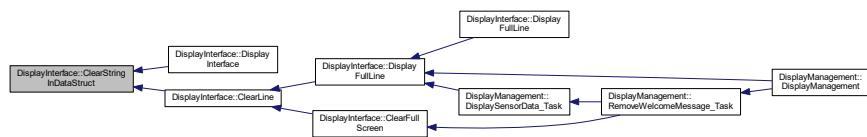
in	<i>line</i>	Line to clear
----	-------------	---------------

## Returns

Nothing

Definition at line 177 of file DisplayInterface.cpp.

Here is the caller graph for this function:



#### 4.7.3.4 DisplayFullLine() [1/2]

```

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

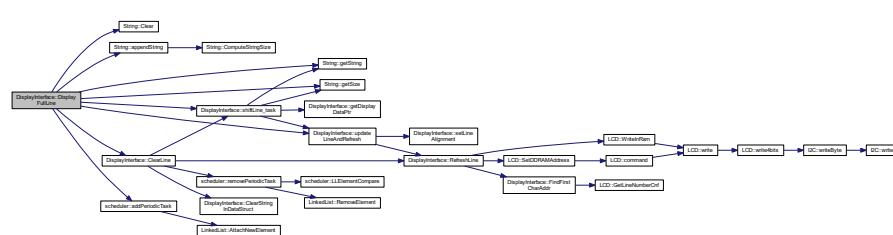
in	<b>str</b>	Pointer to the string to display
in	<b>size</b>	Size of the string to display
in	<b>line</b>	Index of the line where the string shall be displayed
in	<b>mode</b>	Display mode
in	<b>alignment</b>	Requested alignment for the line

#### Returns

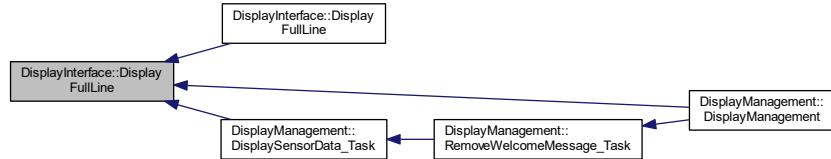
True if the line has been correctly displayed, false otherwise

Definition at line 59 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.7.3.5 DisplayFullLine() [2/2]

```
bool DisplayInterface::DisplayFullLine (
    String * str,
    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

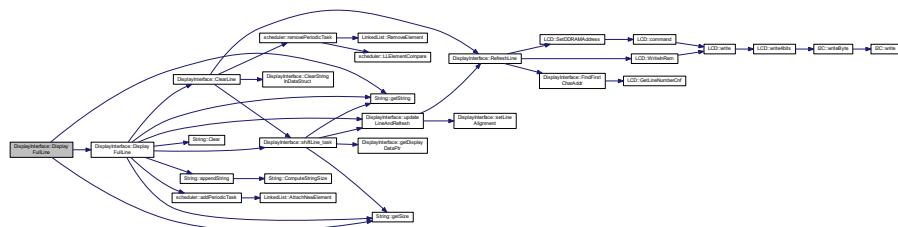
in	<i>str</i>	Pointer to the string to display
in	<i>line</i>	Index of the line where the string shall be displayed
in	<i>mode</i>	Display mode
in	<i>alignment</i>	Requested alignment for the line

##### Returns

True if the line has been correctly displayed, false otherwise

Definition at line 143 of file `DisplayInterface.cpp`.

Here is the call graph for this function:



#### 4.7.3.6 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

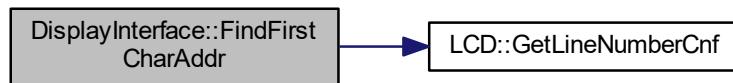
<code>in</code>	<code>line</code>	Line which address shall be found
-----------------	-------------------	-----------------------------------

##### Returns

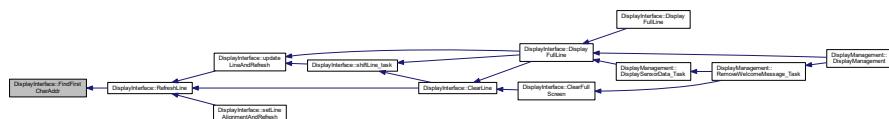
Address in DDRAM of the first character of the line

Definition at line 185 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.7.3.7 getDisplayDataPtr()

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

Display data get function.

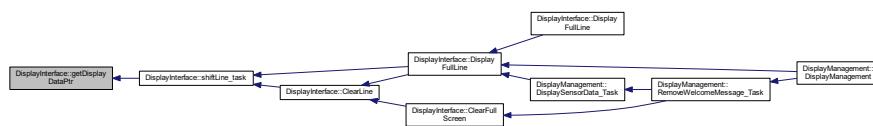
This function returns a pointer to the display data structure.

**Returns**

Pointer to display data structure.

Definition at line 154 of file DisplayInterface.h.

Here is the caller graph for this function:

**4.7.3.8 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 277 of file DisplayInterface.cpp.

**4.7.3.9 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**

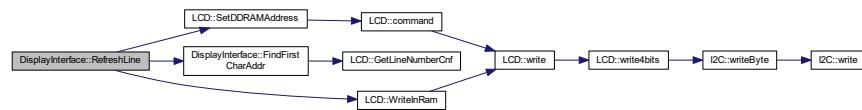
in	<i>line</i>	Line to refresh
----	-------------	-----------------

**Returns**

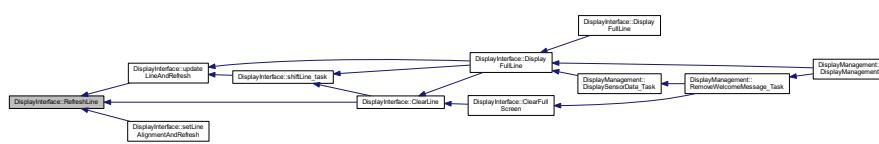
Nothing

Definition at line 164 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.7.3.10 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**

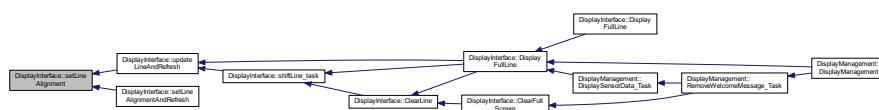
in	<i>line</i>	Line to update
----	-------------	----------------

**Returns**

Nothing

Definition at line 334 of file DisplayInterface.cpp.

Here is the caller graph for this function:



#### 4.7.3.11 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

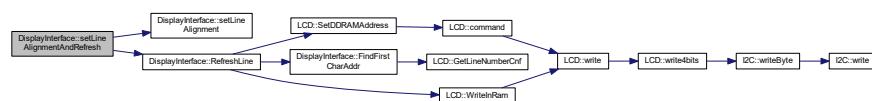
in	<i>line</i>	Requested line to update
in	<i>alignment</i>	Requested alignment for the text

#### Returns

Nothing

Definition at line 452 of file `DisplayInterface.cpp`.

Here is the call graph for this function:



#### 4.7.3.12 shiftLine\_task()

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

Line shifting periodic task.

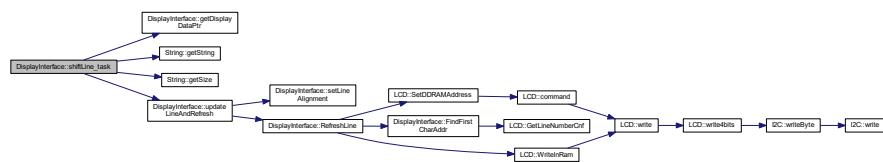
This function is called periodically by the scheduler. It shifts all the lines in line shifting mode and updates the data structures.

**Returns**

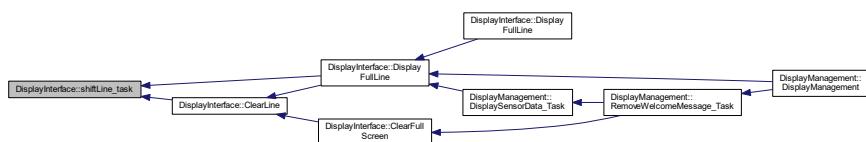
Nothing

Definition at line 286 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.7.3.13 updateLineAndRefresh()**

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

Line data string update function.

This function updates the data string and refreshes the display. It performs a raw update of the line, no processing is done by this function. For calls from outside the class, it is better to use DisplayFullLine function.

**Parameters**

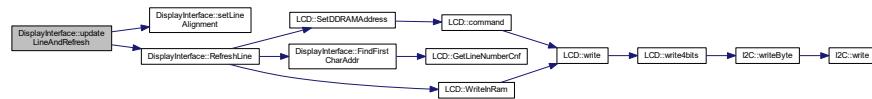
<b>in</b>	<b>str</b>	Pointer to the string to display
<b>in</b>	<b>size</b>	Size of the string
<b>in</b>	<b>line</b>	Line to update

**Returns**

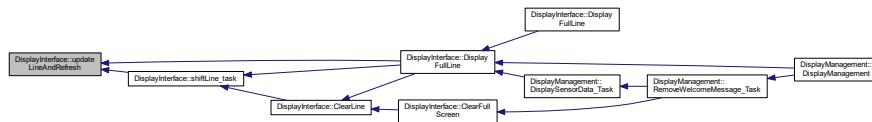
Nothing

Definition at line 148 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.7.4 Member Data Documentation

### 4.7.4.1 display\_data

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

Screen display data

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

### 4.7.4.2 dummy

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

Needed for data alignment

Definition at line 187 of file `DisplayInterface.h`.

### 4.7.4.3 isShiftInProgress

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

Flag indicating if a shift is in progress on any line

Definition at line 189 of file `DisplayInterface.h`.

#### 4.7.4.4 p\_lcd

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

Pointer to the attached LCD driver object

Definition at line 186 of file DisplayInterface.h.

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

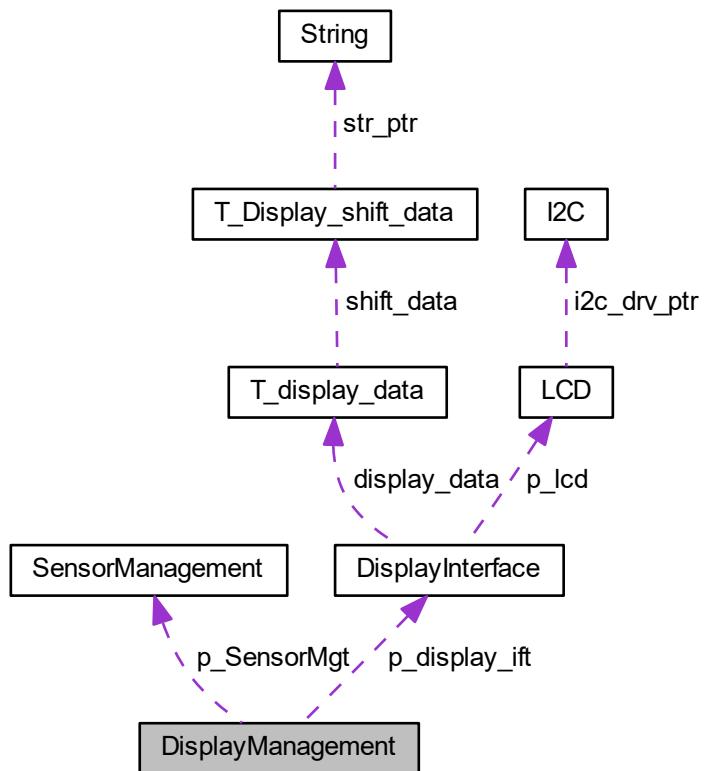
- [DisplayInterface.h](#)
- [DisplayInterface.cpp](#)

## 4.8 DisplayManagement Class Reference

Display management class.

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

Collaboration diagram for DisplayManagement:



## Public Member Functions

- **DisplayManagement ()**  
*Class constructor.*
  - **DisplayInterface \* GetIfItPointer ()**  
*Interface pointer get function.*
  - **SensorManagement \* GetSensorMgtPtr ()**  
*Sensor management 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`
  - `SensorManagement * p_SensorMgt`

#### 4.8.1 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 47 of file DisplayManagement.h.

#### 4.8.2 Constructor & Destructor Documentation

#### 4.8.2.1 DisplayManagement()

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

Class constructor.

This class initializes display management.

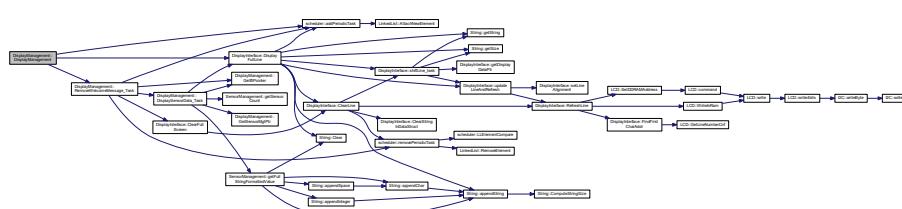
It created a display interface object and initializes all class variables.

## Returns

Nothing

Definition at line 30 of file DisplayManagement.cpp.

Here is the call graph for this function:



#### 4.8.3 Member Function Documentation

#### 4.8.3.1 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.

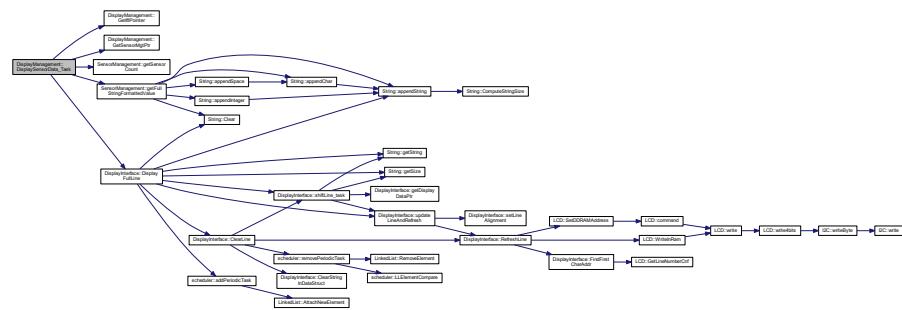
It is called periodically by scheduler.

## Returns

Nothing

Definition at line 71 of file DisplayManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.8.3.2 GetIfPointer()

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

Interface pointer get function.

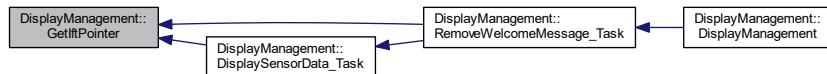
This function returns the pointer to the display interface object

##### Returns

Pointer to display interface object

Definition at line 76 of file DisplayManagement.h.

Here is the caller graph for this function:



#### 4.8.3.3 GetSensorMgtPtr()

```
SensorManagement* DisplayManagement::GetSensorMgtPtr ( ) [inline]
```

**Sensor** management pointer get function.

This function returns the pointer to the sensor management object

##### Returns

Pointer to sensor management object

Definition at line 87 of file DisplayManagement.h.

Here is the caller graph for this function:



#### 4.8.3.4 RemoveWelcomeMessage\_Task()

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

End of welcome message task.

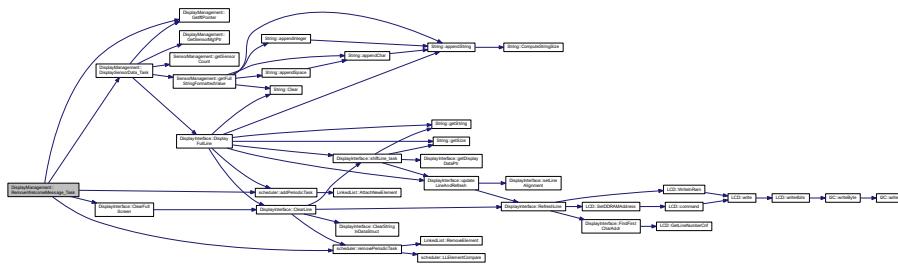
This task clears the welcome message from the screen and start periodic display of sensor data. This task shall be added in scheduler when the welcome message is displayed on screen. As it shall be called only once, the task removes itself from the scheduler after the first call.

##### Returns

Nothing

Definition at line 54 of file DisplayManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.8.4 Member Data Documentation

##### 4.8.4.1 p\_display\_ift

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

Pointer to the display interface object

Definition at line 104 of file DisplayManagement.h.

#### 4.8.4.2 p\_SensorMgt

```
SensorManagement* DisplayManagement::p_SensorMgt [private]
```

Pointer to the sensor management object

Definition at line 105 of file DisplayManagement.h.

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

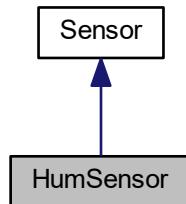
- [DisplayManagement.h](#)
- [DisplayManagement.cpp](#)

## 4.9 HumSensor Class Reference

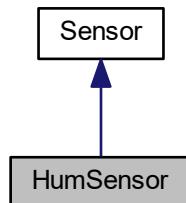
Class for humidity sensor.

```
#include <HumSensor.h>
```

Inheritance diagram for HumSensor:



Collaboration diagram for HumSensor:



## Public Member Functions

- [HumSensor \(\)](#)  
*Class constructor.*
- [HumSensor \(uint16\\_t val\\_tmo, uint16\\_t period\)](#)  
*Overloaded class constructor.*
- [bool updateTaskPeriod \(uint16\\_t period\)](#)  
*Task period update.*

## Static Public Member Functions

- [static void readHumSensor\\_task \(\)](#)  
*Task for reading humidity values.*

## Additional Inherited Members

### 4.9.1 Detailed Description

Class for humidity sensor.

This class defines all functions used to read data from humidity sensor and monitor it. It is inherited from class [Sensor](#).

Definition at line 18 of file [HumSensor.h](#).

### 4.9.2 Constructor & Destructor Documentation

#### 4.9.2.1 [HumSensor\(\)](#) [1/2]

```
HumSensor::HumSensor ( )
```

Class constructor.

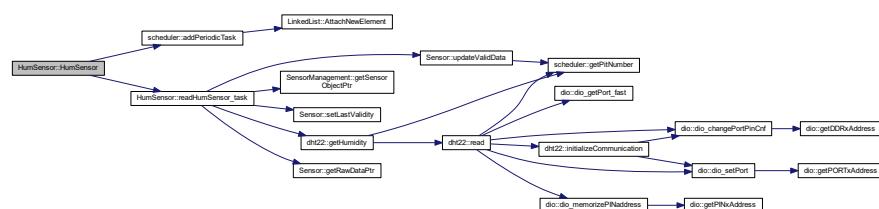
This function initializes all data of the class [HumSensor](#). If needed, it creates a new instance of the DHT22 sensor object. It also adds the periodic task in the scheduler.

#### Returns

Nothing

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

Here is the call graph for this function:



#### 4.9.2.2 HumSensor() [2/2]

```
HumSensor::HumSensor (
    uint16_t val_tmo,
    uint16_t period )
```

Overloaded class constructor.

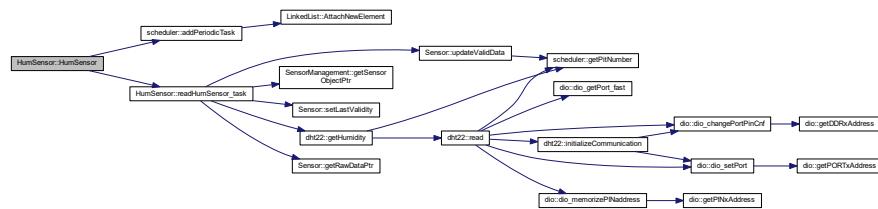
This function initializes all data of the class `HumSensor`. It sets validity timeout and task period to the given value. If needed, it creates a new instance of the DHT22 sensor object. It also adds the periodic task in the scheduler.

#### Returns

Nothing

Definition at line 38 of file `HumSensor.cpp`.

Here is the call graph for this function:



### 4.9.3 Member Function Documentation

#### 4.9.3.1 readHumSensor\_task()

```
void HumSensor::readHumSensor_task ( ) [static]
```

Task for reading humidity values.

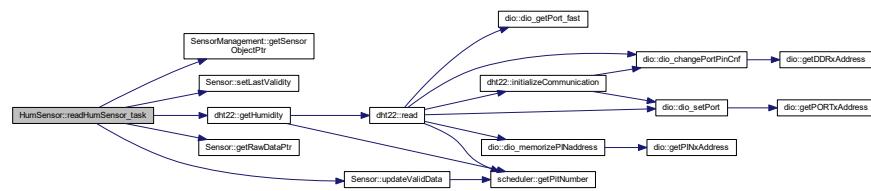
This task reads humidity data using DHT22 driver. It is called periodically.

**Returns**

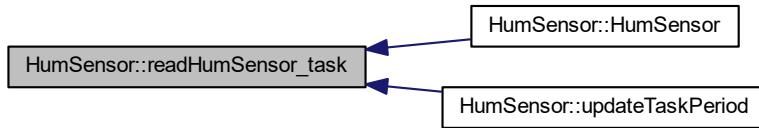
Nothing

Definition at line 48 of file HumSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.9.3.2 updateTaskPeriod()**

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

Task period update.

This function updates the period of the temperature task.

**Parameters**

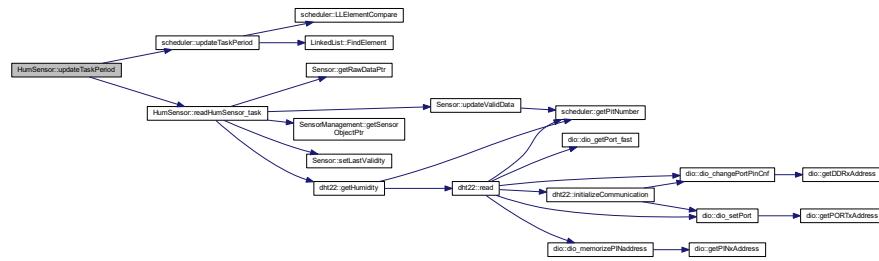
in	<i>period</i>	New period of the task
----	---------------	------------------------

**Returns**

True if the period has been updated, false otherwise

Definition at line 62 of file HumSensor.cpp.

Here is the call graph for this function:



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

- HumSensor.h
  - HumSensor.cpp

## 4.10 I2C Class Reference

Two-wire serial interface ([I<sub>2</sub>C](#)) class definition.

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

## Public Member Functions

- **I2C** (`uint32_t l_bitrate`)  
*I2C class constructor.*
  - bool **writeByte** (`uint8_t *data, uint8_t tx_address`)  
*Byte sending function.*
  - bool **write** (`uint8_t *data, uint8_t tx_address, uint8_t size`)  
*I2C write function.*
  - void **setBitRate** (`uint32_t l_bitrate`)  
*Variable bitrate setting function.*

## Private Member Functions

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

## Private Attributes

- `uint32_t bitrate`

### 4.10.1 Detailed Description

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

This class manages [I2C](#) driver.

Definition at line 23 of file I2C.h.

### 4.10.2 Constructor & Destructor Documentation

#### 4.10.2.1 I2C()

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

[I2C](#) class constructor.

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

##### Parameters

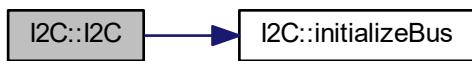
in	<i>l_bitrate</i>	Requested bitrate for <a href="#">I2C</a> bus (in Hz)
----	------------------	---

##### Returns

Nothing

Definition at line 16 of file I2C.cpp.

Here is the call graph for this function:



### 4.10.3 Member Function Documentation

#### 4.10.3.1 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 79 of file I2C.cpp.

Here is the caller graph for this function:



#### 4.10.3.2 setBitRate()

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

Variable bitrate setting function.

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

##### Parameters

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

##### Returns

Nothing

Definition at line 74 of file I2C.cpp.

### 4.10.3.3 write()

```
    bool I2C::write (
```

## I2C write function.

This function sends the requested number of bytes to the I<sub>2</sub>C device with the given address

## Parameters

in	<i>data</i>	Pointer to the data to send
in	<i>tx_address</i>	I2C address of the device
in	<i>size</i>	Number of bytes to send

## Returns

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

Definition at line 28 of file I2C.cpp.

Here is the caller graph for this function:



#### 4.10.3.4 writeByte()

```
bool I2C::writeByte (
```

Byte sending function.

This function sends one byte to the [I2C](#) device with the given address. It only calls "write" function with size equal to 1. Kept for compatibility with [LCD](#) driver.

## Parameters

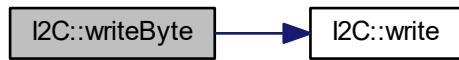
in	<i>data</i>	Pointer to the data to send
in	<i>tx_address</i>	I <sub>2</sub> C address of the device

## Returns

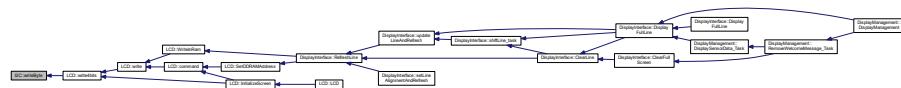
True if transmission is completed, False if an error has occurred

Definition at line 23 of file I2C.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### **4.10.4 Member Data Documentation**

#### 4.10.4.1 bitrate

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

Definition at line 66 of file I2C.h.

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

- I2C.h
  - I2C.cpp

## 4.11 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.*

### 4.11.1 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.

### 4.11.2 Constructor & Destructor Documentation

#### 4.11.2.1 keepAliveLed()

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

Class constructor.

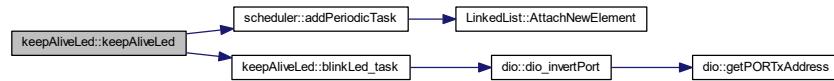
This function initializes the class keepAliveLed

#### Returns

Nothing

Definition at line 22 of file keepAliveLed.cpp.

Here is the call graph for this function:



### 4.11.3 Member Function Documentation

#### 4.11.3.1 `blinkLed_task()`

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

Task for LED blinking.

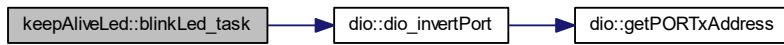
This function is inserted into the scheduler. It changes the state of the LED output to make it blink

##### 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:

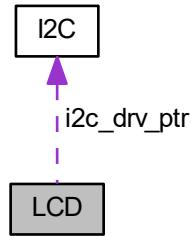
- [keepAliveLed.h](#)
- [keepAliveLed.cpp](#)

## 4.12 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`

### 4.12.1 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.

### 4.12.2 Constructor & Destructor Documentation

#### 4.12.2.1 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

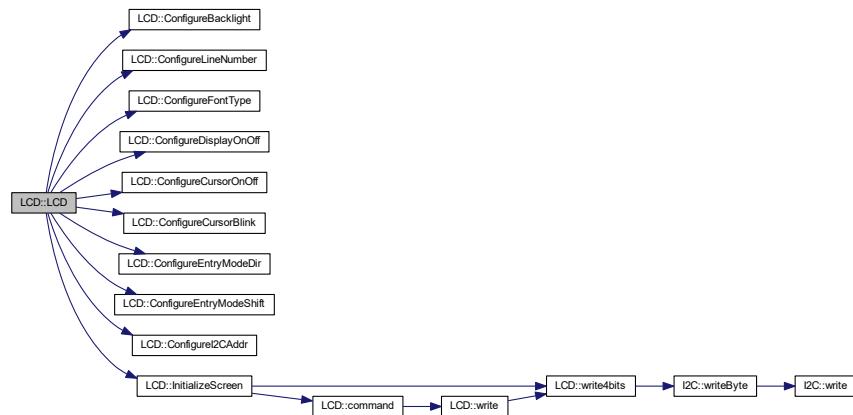
in	<code>init_conf</code>	Initial configuration structure
----	------------------------	---------------------------------

**Returns**

Nothing

Definition at line 18 of file LCD.cpp.

Here is the call graph for this function:



### 4.12.3 Member Function Documentation

#### 4.12.3.1 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**

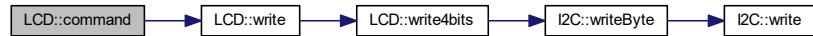
in	cmd	Requested command
----	-----	-------------------

**Returns**

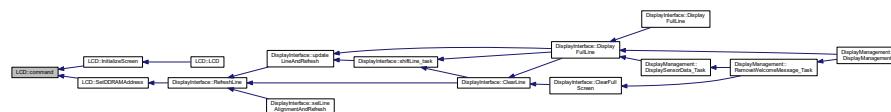
Nothing

Definition at line 123 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.12.3.2 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	<code>enable</code>	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:



#### 4.12.3.3 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:



#### 4.12.3.4 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:



#### 4.12.3.5 ConfigureDisplayOnOff()

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

Display configuration function.

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

##### Parameters

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

##### Returns

Nothing

Definition at line 214 of file LCD.h.

Here is the caller graph for this function:



#### 4.12.3.6 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:

**4.12.3.7 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:



#### 4.12.3.8 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:



#### 4.12.3.9 ConfigureI2CAddr()

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

I2C address configuration function.

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

##### Parameters

in	param	I2C address
----	-------	-------------

##### Returns

Nothing

Definition at line 274 of file LCD.h.

Here is the caller graph for this function:



#### 4.12.3.10 ConfigureLineNumber()

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

Line type configuration function.

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

##### Parameters

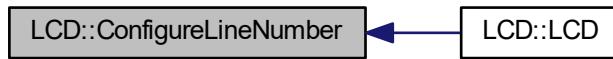
in	param	Configuration value
----	-------	---------------------

##### Returns

Nothing

Definition at line 190 of file LCD.h.

Here is the caller graph for this function:



#### 4.12.3.11 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.

#### 4.12.3.12 GetLineNumberCnf()

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

Number of line get function.

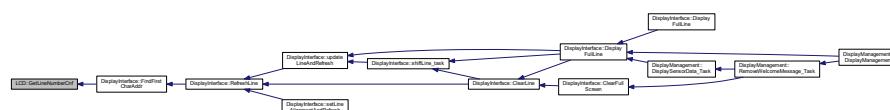
This function returns the line number configuration of the screen : 1 or 2 lines mode.

##### Returns

Line number configuration

Definition at line 316 of file LCD.h.

Here is the caller graph for this function:



#### 4.12.3.13 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:



#### 4.12.3.14 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

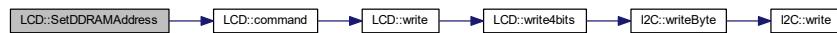
in	<i>addr</i>	New DDRAM address
----	-------------	-------------------

**Returns**

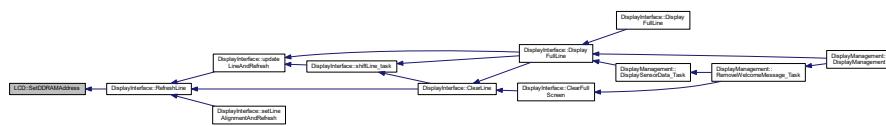
Nothing

Definition at line 166 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.12.3.15 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 <a href="#">LCD</a> communication

**Returns**

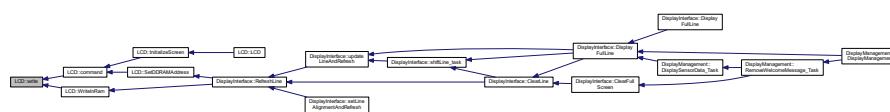
Nothing

Definition at line 62 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.12.3.16 write4bits()

```
void LCD::write4bits (
```

### I2C write function for 4-bits mode.

This function sends the requested 8-bits data on the I<sub>C</sub> 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

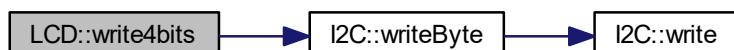
in	<i>data</i>	8-bit data to send
----	-------------	--------------------

## Returns

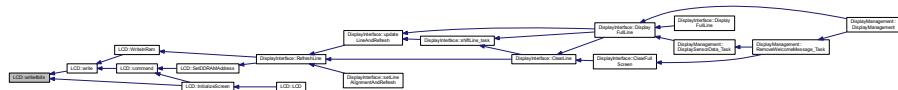
Nothing

Definition at line 45 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.12.3.17 WriteInRam()

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

Screen RAM write function.

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

##### Parameters

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

##### Returns

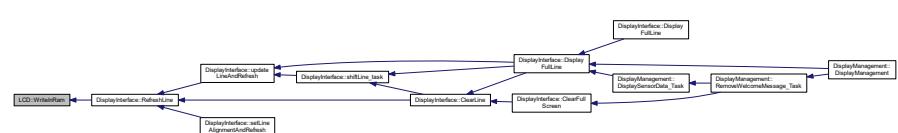
Nothing

Definition at line 188 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.12.4 Member Data Documentation

##### 4.12.4.1 `backlight_enable`

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

Backlight enable flag

Definition at line 324 of file LCD.h.

##### 4.12.4.2 `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.

##### 4.12.4.3 `cnfCursorOnOff`

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

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

Definition at line 328 of file LCD.h.

##### 4.12.4.4 `cnfDisplayOnOff`

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

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

Definition at line 327 of file LCD.h.

##### 4.12.4.5 `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.

#### 4.12.4.6 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.

#### 4.12.4.7 cnfFontType

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

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

Definition at line 326 of file LCD.h.

#### 4.12.4.8 cnfI2C\_addr

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

I2C address of the [LCD](#) screen

Definition at line 332 of file LCD.h.

#### 4.12.4.9 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.

#### 4.12.4.10 ddram\_addr

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

Screen DDRAM address

Definition at line 336 of file LCD.h.

## 4.12.4.11 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:

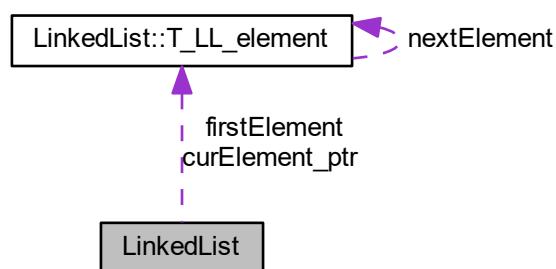
- [LCD.h](#)
- [LCD.cpp](#)

## 4.13 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](#)

### 4.13.1 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](#).

### 4.13.2 Member Typedef Documentation

#### 4.13.2.1 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.

### 4.13.3 Constructor & Destructor Documentation

#### 4.13.3.1 LinkedList()

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

Class constructor.

This constructor initializes a linked list

Returns

Nothing

Definition at line 18 of file LinkedList.cpp.

#### 4.13.3.2 ~LinkedList()

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

Class destructor.

This function deletes the linked list

Returns

Nothing

Definition at line 24 of file LinkedList.cpp.

Here is the call graph for this function:



## 4.13.4 Member Function Documentation

### 4.13.4.1 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

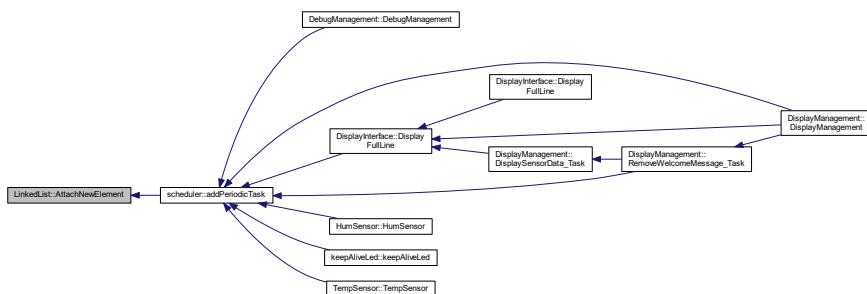
in	<i>data_ptr</i>	Pointer to the data element
----	-----------------	-----------------------------

#### Returns

Nothing

Definition at line 42 of file LinkedList.cpp.

Here is the caller graph for this function:



### 4.13.4.2 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**

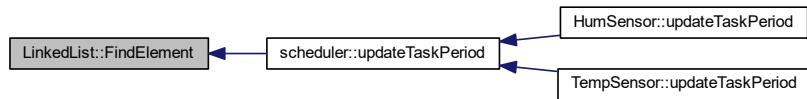
in	<i>comparisonFct_ptr</i>	Pointer to the comparison function
in	<i>reference_ptr</i>	Pointer to the element to find in the chain
out	<i>chainElement_ptr</i>	Pointer to pointer to the found element

**Returns**

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

Definition at line 133 of file LinkedList.cpp.

Here is the caller graph for this function:

**4.13.4.3 getCurrentElement()**

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

Current element get function.

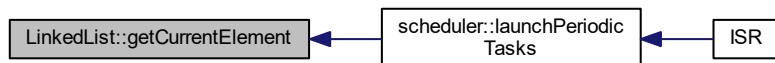
This function returns a pointer to the current pointed data in the chain.

**Returns**

Pointer to the current data

Definition at line 67 of file LinkedList.h.

Here is the caller graph for this function:



#### 4.13.4.4 IsLLEmpty()

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

Empty linked list.

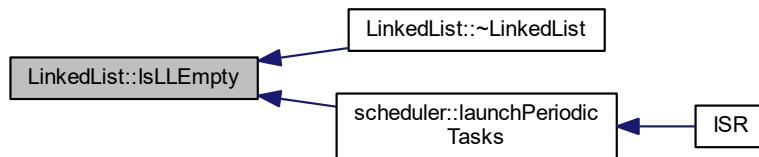
This function checks whether the linked list is empty or not (pointer to first element is equal to 0 or not).

##### Returns

True if the list is empty, false otherwise

Definition at line 125 of file LinkedList.cpp.

Here is the caller graph for this function:



#### 4.13.4.5 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 111 of file LinkedList.cpp.

Here is the caller graph for this function:



#### 4.13.4.6 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

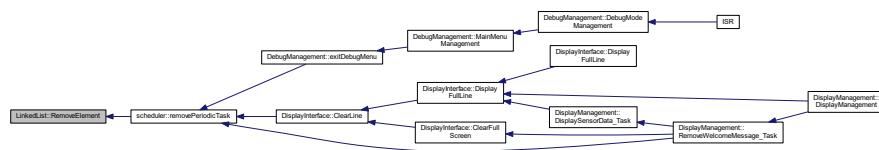
in	<i>comparisonFct_ptr</i>	Pointer to the comparison function to use
in	<i>reference_ptr</i>	Pointer to the reference data used for comparison

#### Returns

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

Definition at line 67 of file LinkedList.cpp.

Here is the caller graph for this function:



#### 4.13.4.7 ResetElementPtr()

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

Resets element pointer.

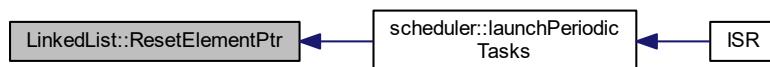
This function sets the element pointer to the first element of the chain.

#### Returns

Nothing

Definition at line 86 of file LinkedList.h.

Here is the caller graph for this function:



### 4.13.5 Member Data Documentation

#### 4.13.5.1 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`.

#### 4.13.5.2 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:

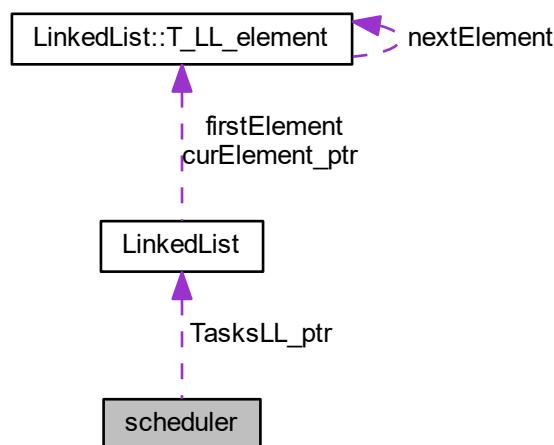
- [LinkedList.h](#)
- [LinkedList.cpp](#)

## 4.14 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](#)

### 4.14.1 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.

## 4.14.2 Member Typedef Documentation

### 4.14.2.1 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.

## 4.14.3 Constructor & Destructor Documentation

### 4.14.3.1 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:



## 4.14.4 Member Function Documentation

### 4.14.4.1 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**

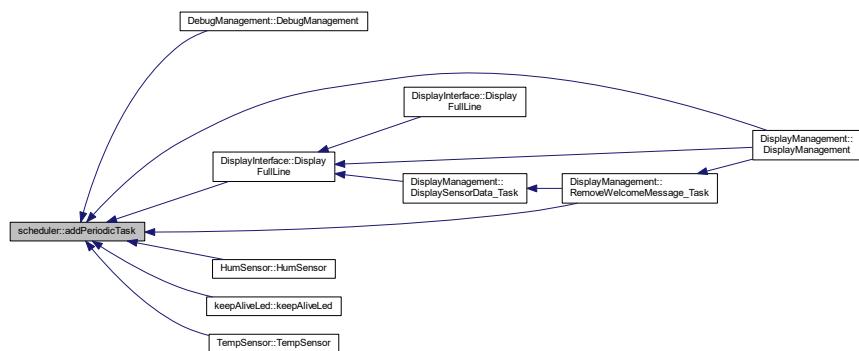
Nothing

Definition at line 99 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.14.4.2 getPitNumber()**

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

Get function for PIT number.

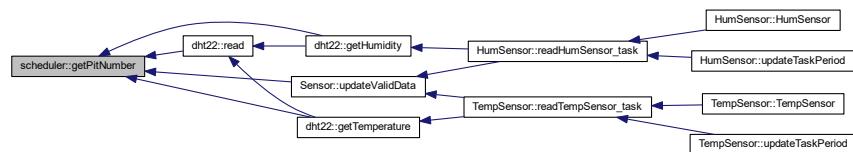
This function returns the PIT number

**Returns**

PIT number

Definition at line 113 of file scheduler.cpp.

Here is the caller graph for this function:

**4.14.4.3 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.

**4.14.4.4 launchPeriodicTasks()**

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

Main scheduler function.

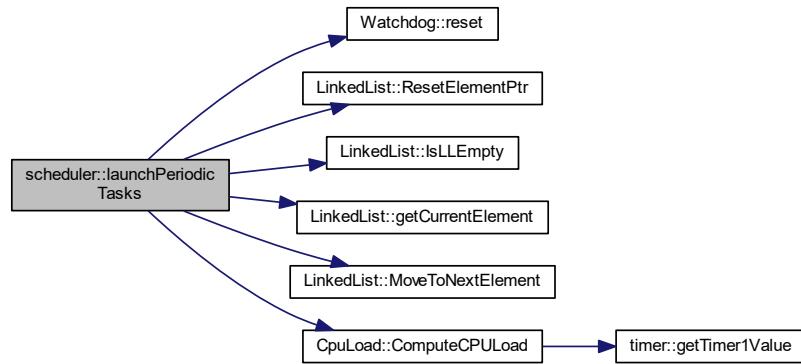
This function launches the scheduled tasks according to current software time and task configuration

Returns

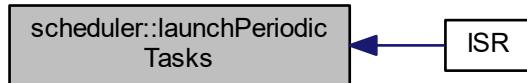
Nothing

Definition at line 54 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.14.4.5 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**

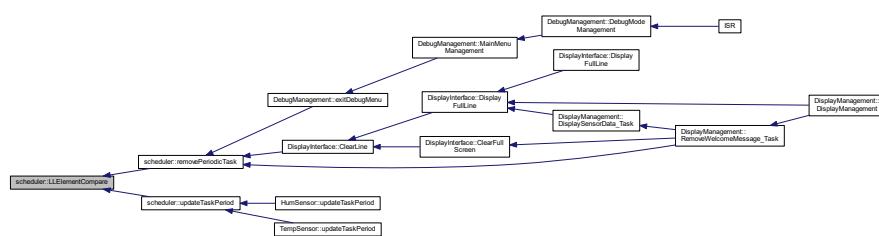
in	<i>LLElement</i>	Pointer to the linked list element
in	<i>CompareElement</i>	Pointer to the element to the compare

**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:

**4.14.4.6 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**

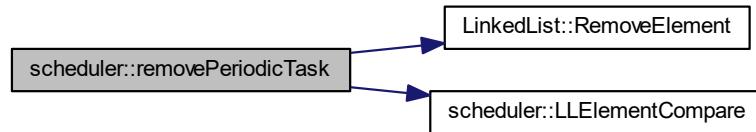
in	<i>task_ptr</i>	address of the task to remove from scheduler
----	-----------------	--

**Returns**

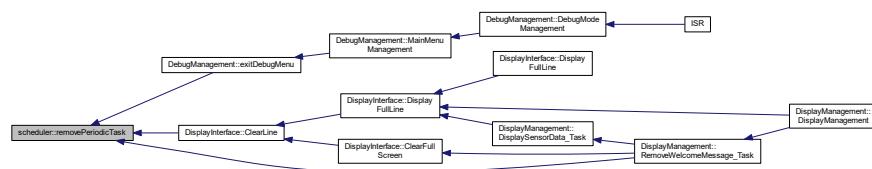
TRUE if the task has been removed, FALSE if the task does not exist in the scheduler

Definition at line 119 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.14.4.7 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:



#### 4.14.4.8 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

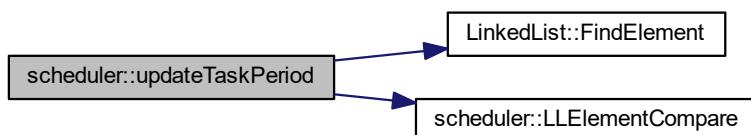
in	<i>task_ptr</i>	Pointer of the task to update
in	<i>period</i>	New period of the task

##### Returns

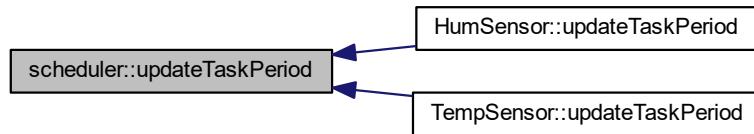
True if the update has been correctly done, false otherwise

Definition at line 142 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.14.5 Member Data Documentation

##### 4.14.5.1 pit\_number

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

Counter of periodic interrupts

Definition at line 140 of file `scheduler.h`.

##### 4.14.5.2 task\_count

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

Number of task in scheduler

Definition at line 136 of file `scheduler.h`.

##### 4.14.5.3 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:

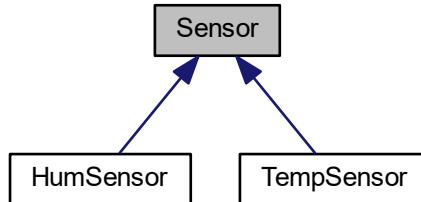
- [scheduler.h](#)
- [scheduler.cpp](#)

## 4.15 Sensor Class Reference

Generic class for sensor device.

```
#include <Sensor.h>
```

Inheritance diagram for Sensor:



### Public Member Functions

- `Sensor ()`  
*Sensor class constructor.*
- `Sensor (uint16_t val_tmo, uint16_t period)`  
*Overloaded sensor class constructor.*
- `uint16_t * getRawDataPtr ()`  
*Get pointer to raw sensor data.*
- `bool getValue (uint16_t *value)`  
*Get sensor value function.*
- `void setLastValidity (bool validity)`  
*Validity setting function.*
- `void updateValidData ()`
- `uint16_t getValueInteger ()`  
*Data formatting function - Integer part.*
- `uint8_t getValueDecimal ()`  
*Data 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.*
- `void setValidityTMO (uint16_t timeout)`  
*Validity timeout setting function.*

### Static Public Member Functions

- `static void readSensor_task ()`  
*Task for reading sensor values.*

## Protected Attributes

- bool `validity`
- bool `validity_last_read`
- uint32\_t `valid_pit`
- uint16\_t `validity_tmo`
- uint16\_t `raw_data`
- uint16\_t `valid_value`
- uint16\_t `task_period`

### 4.15.1 Detailed Description

Generic class for sensor device.

This class defines a generic sensor, as handled by class [SensorManagement](#). It should not be instantiated. Only inherited classes shall be instantiated.

Definition at line 18 of file Sensor.h.

### 4.15.2 Constructor & Destructor Documentation

#### 4.15.2.1 `Sensor()` [1/2]

```
Sensor::Sensor ( )
```

`Sensor` class constructor.

This function initializes the class.

**Returns**

Nothing

Definition at line 22 of file Sensor.cpp.

#### 4.15.2.2 `Sensor()` [2/2]

```
Sensor::Sensor (
    uint16_t val_tmo,
    uint16_t period )
```

Overloaded sensor class constructor.

This function initializes the class. It sets validity timeout and task period to the given value.

**Parameters**

in	<i>val_tmo</i>	Validity timeout
in	<i>task_period</i>	Task period

**Returns**

Nothing

Definition at line 38 of file Sensor.cpp.

### 4.15.3 Member Function Documentation

#### 4.15.3.1 getRawDataPtr()

```
uint16_t* Sensor::getRawDataPtr ( ) [inline]
```

Get pointer to raw sensor data.

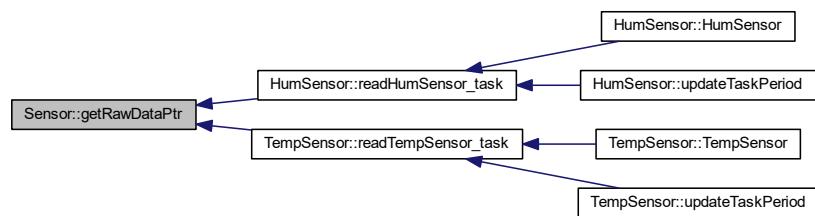
This function returns a pointer to the class member `raw_data`

**Returns**

Pointer to `raw_data`

Definition at line 53 of file Sensor.h.

Here is the caller graph for this function:



#### 4.15.3.2 getTaskPeriod()

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

Task period get function.

This function returns the period of the sensor task

##### Returns

Period of the task (ms)

Definition at line 137 of file Sensor.h.

#### 4.15.3.3 getValidity()

```
bool Sensor::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 117 of file Sensor.h.

#### 4.15.3.4 getValue()

```
bool Sensor::getValue (
    uint16_t * value ) [inline]
```

Get sensor value function.

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

##### Parameters

out	value	Sensor value
-----	-------	--------------

##### Returns

Validity

Definition at line 64 of file Sensor.h.

#### 4.15.3.5 `getValueDecimal()`

```
uint8_t Sensor::getValueDecimal ( ) [inline]
```

Data formatting function - Decimal part.

This function return the decimal part of the sensor value

##### Returns

Decimal value of the sensor data

Definition at line 106 of file Sensor.h.

#### 4.15.3.6 `getValueInteger()`

```
uint16_t Sensor::getValueInteger ( ) [inline]
```

Data formatting function - Integer part.

This function return the integer part of the sensor value

##### Returns

Integer value of the sensor data

Definition at line 95 of file Sensor.h.

#### 4.15.3.7 `readSensor_task()`

```
static void Sensor::readSensor_task ( ) [inline], [static]
```

Task for reading sensor values.

This task reads sensor data using sensor driver. It is called periodically. This function shall be re-written in each inherited class.

##### Returns

Nothing

Definition at line 46 of file Sensor.h.

#### 4.15.3.8 `setLastValidity()`

```
void Sensor::setLastValidity (
    bool validity ) [inline]
```

Validity setting function.

This function sets the class member `validity_last_read`

**Parameters**

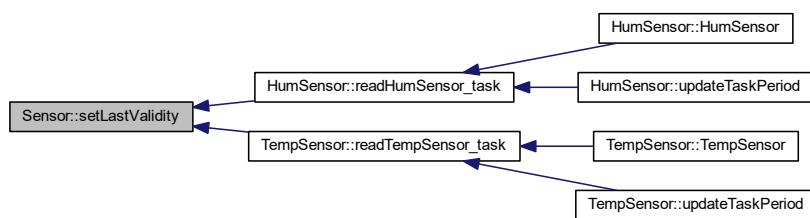
in	<i>validity</i>	Value of validity
----	-----------------	-------------------

**Returns**

Nothing

Definition at line 76 of file Sensor.h.

Here is the caller graph for this function:

**4.15.3.9 setValidityTMO()**

```
void Sensor::setValidityTMO (
    uint16_t timeout ) [inline]
```

Validity timeout setting function.

This function sets the validity timeout.

**Parameters**

in	<i>timeout</i>	New value of timeout.
----	----------------	-----------------------

**Returns**

Nothing

Definition at line 149 of file Sensor.h.

#### 4.15.3.10 updateTaskPeriod()

```
bool Sensor::updateTaskPeriod (
    uint16_t period ) [inline]
```

Task period update.

This function updates the period of the sensor task. It shall be re-written in each inherited class.

##### Parameters

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

##### Returns

True if the period has been updated, false otherwise

Definition at line 129 of file Sensor.h.

#### 4.15.3.11 updateValidData()

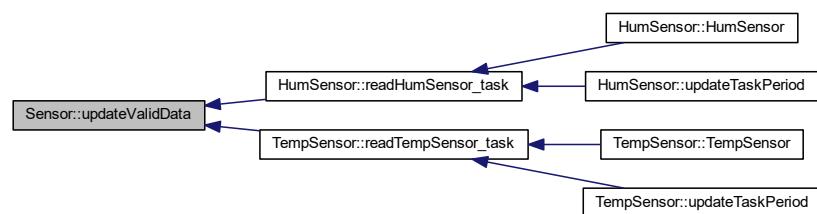
```
void Sensor::updateValidData ( )
```

Definition at line 53 of file Sensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.15.4 Member Data Documentation

##### 4.15.4.1 raw\_data

```
uint16_t Sensor::raw_data [protected]
```

Raw value of sensor data (directly coming from driver)

Definition at line 161 of file Sensor.h.

##### 4.15.4.2 task\_period

```
uint16_t Sensor::task_period [protected]
```

Task period

Definition at line 165 of file Sensor.h.

##### 4.15.4.3 valid坑

```
uint32_t Sensor::valid坑 [protected]
```

pit number of the last time when data were valid

Definition at line 158 of file Sensor.h.

##### 4.15.4.4 valid\_value

```
uint16_t Sensor::valid_value [protected]
```

Valid value of sensor data

Definition at line 163 of file Sensor.h.

##### 4.15.4.5 validity

```
bool Sensor::validity [protected]
```

Validity of sensor data

Definition at line 155 of file Sensor.h.

#### 4.15.4.6 validity\_last\_read

```
bool Sensor::validity_last_read [protected]
```

Validity of last read sensor data

Definition at line 156 of file Sensor.h.

#### 4.15.4.7 validity\_tmo

```
uint16_t Sensor::validity_tmo [protected]
```

Number of PITs after which the sensor value is declared invalid

Definition at line 159 of file Sensor.h.

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

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

## 4.16 SensorManagement Class Reference

[Sensor](#) management class.

```
#include <SensorManagement.h>
```

### Public Member Functions

- [SensorManagement \(\)](#)  
*Class constructor.*
- [uint8\\_t getSensorCount \(\)](#)  
*Sensor count get function.*
- [bool updateTaskPeriod \(uint16\\_t period\)](#)  
*Sensors tasks period update.*
- [void getFullStringFormattedValue \(uint8\\_t sensor\\_idx, String \\*str\)](#)  
*Sensor value formatting function.*
- [void \\* getSensorObjectPtr \(T\\_SensorManagement\\_Sensor\\_Type type\)](#)  
*Sensor object pointer get function.*

### Private Attributes

- [uint8\\_t nb\\_sensors](#)
- [void \\*\\* sensor\\_ptr\\_table](#)

### 4.16.1 Detailed Description

**Sensor** management class.

This class manages all sensors present in the SW. It manages sensor activation and deactivation, has a periodic task to retrieve sensor data. It also creates the string with sensors values used by display services.

Definition at line 29 of file SensorManagement.h.

### 4.16.2 Constructor & Destructor Documentation

#### 4.16.2.1 SensorManagement()

```
SensorManagement::SensorManagement ( )
```

Class constructor.

This function initializes the class. It allocates the sensor pointer table according to the number of sensors present. For each sensor, the related object is created.

**Returns**

Nothing

Definition at line 26 of file SensorManagement.cpp.

### 4.16.3 Member Function Documentation

#### 4.16.3.1 getFullStringFormattedValue()

```
void SensorManagement::getFullStringFormattedValue (
    uint8_t sensor_idx,
    String * str )
```

**Sensor** value formatting function.

This function gets the value of the selected sensor and formats it into a string using the data name string defined in the configuration.

**Parameters**

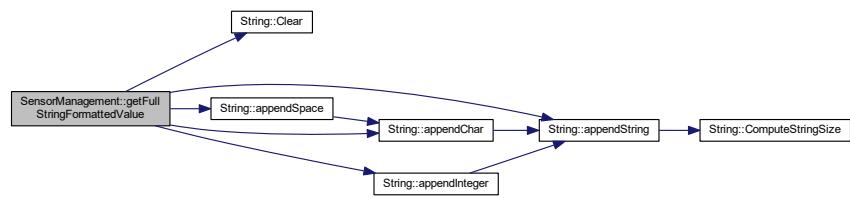
in	<i>sensor_idx</i>	Index of the requested sensor
out	<i>str</i>	Pointer to the formatted string

**Returns**

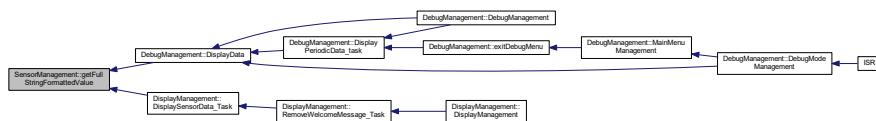
Nothing

Definition at line 70 of file SensorManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.16.3.2 getSensorCount()

```
uint8_t SensorManagement::getSensorCount( ) [inline]
```

**Sensor** count get function.

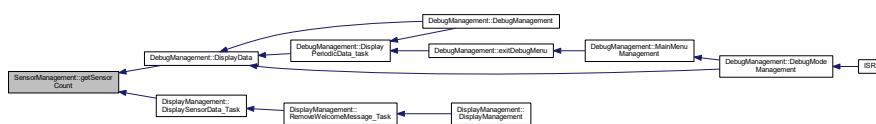
This function returns the number of sensors present in the SW

**Returns**

Number of sensors.

Definition at line 47 of file SensorManagement.h.

Here is the caller graph for this function:



#### 4.16.3.3 getSensorObjectPtr()

```
void * SensorManagement::getSensorObjectPtr (
    T_SensorManagement_Sensor_Type type )
```

**Sensor** object pointer get function.

This function finds the pointer to the sensor object of the given type in `sensor_ptr_table` and returns this pointer.

**Parameters**

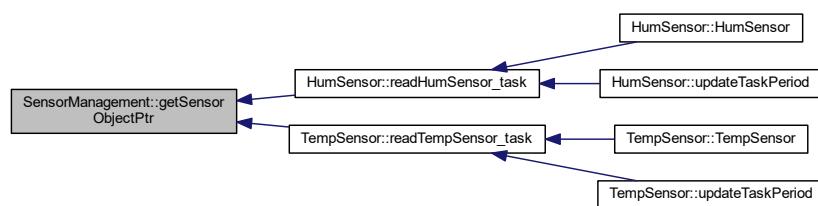
in	type	Type of sensor to find
----	------	------------------------

**Returns**

Pointer to the sensor object (casted to pointer-to-void)

Definition at line 90 of file SensorManagement.cpp.

Here is the caller graph for this function:

**4.16.3.4 updateTaskPeriod()**

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

Sensors tasks period update.

This function updates the period of all sensors tasks. The function updateTaskPeriod is called for each sensor object.

**Parameters**

in	period	New period.
----	--------	-------------

**Returns**

True if the period has been updated, false otherwise.

Definition at line 53 of file SensorManagement.cpp.

**4.16.4 Member Data Documentation**

#### 4.16.4.1 nb\_sensors

```
uint8_t SensorManagement::nb_sensors [private]
```

Number of sensors

Definition at line 82 of file SensorManagement.h.

#### 4.16.4.2 sensor\_ptr\_table

```
void** SensorManagement::sensor_ptr_table [private]
```

Table containing pointers to all sensors objects (declared as pointer to void to avoid including [Sensor.h](#) in all files)

Definition at line 83 of file SensorManagement.h.

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

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

## 4.17 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.*
- [void appendSpace \(\)](#)  
*Space adding function.*

## Private Member Functions

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

## Private Attributes

- `uint8_t * string`
- `uint8_t size`

### 4.17.1 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.

### 4.17.2 Constructor & Destructor Documentation

#### 4.17.2.1 `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

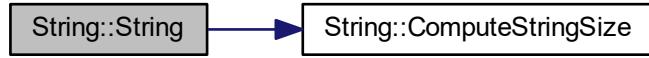
in	<code>str</code>	Pointer to initialization string
----	------------------	----------------------------------

#### Returns

Nothing

Definition at line 15 of file String.cpp.

Here is the call graph for this function:



#### 4.17.2.2 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.

#### 4.17.2.3 ~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:



### 4.17.3 Member Function Documentation

#### 4.17.3.1 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

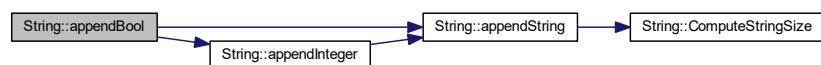
in	<code>data</code>	Boolean data to add
in	<code>isText</code>	Defines the conversion mode : text or integer

##### 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:



### 4.17.3.2 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

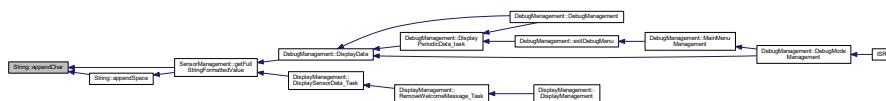
Nothing

Definition at line 139 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



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

## Returns

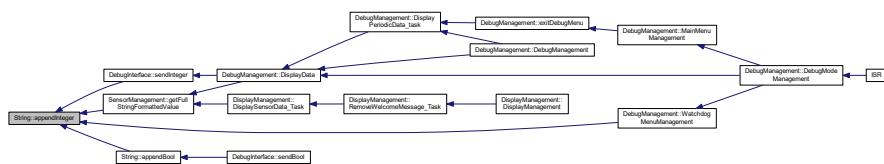
Nothing

Definition at line 95 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.17.3.4 appendSpace()

```
void String::appendSpace ( ) [inline]
```

Space adding function.

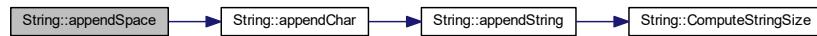
This function adds a space at the end of the string. It only calls appendChar function.

**Returns**

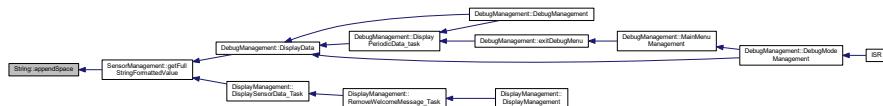
Nothing

Definition at line 123 of file String.h.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.17.3.5 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**

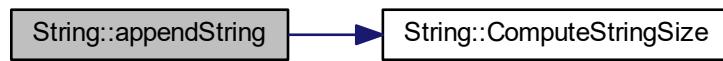
in	<i>str</i>	New string to add
----	------------	-------------------

**Returns**

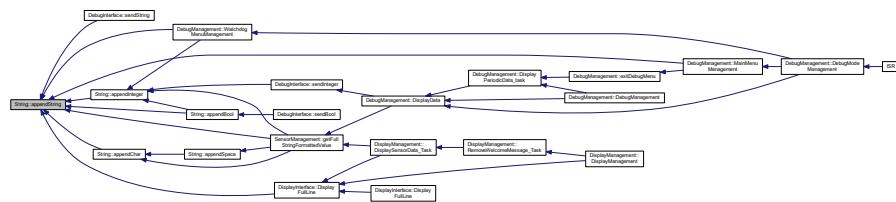
Nothing

Definition at line 57 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.17.3.6 Clear()**

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

`String` clear function.

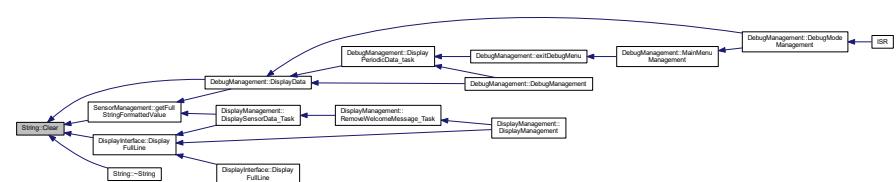
This function clears the string. Size is set to 0 and the memory is freed.

**Returns**

Nothing

Definition at line 113 of file String.cpp.

Here is the caller graph for this function:



#### 4.17.3.7 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

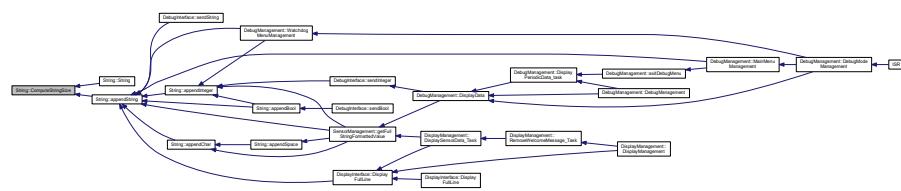
in	<i>str</i>	Pointer to the beginning of the string
----	------------	--

#### 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:



#### 4.17.3.8 getSize()

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

Size get function.

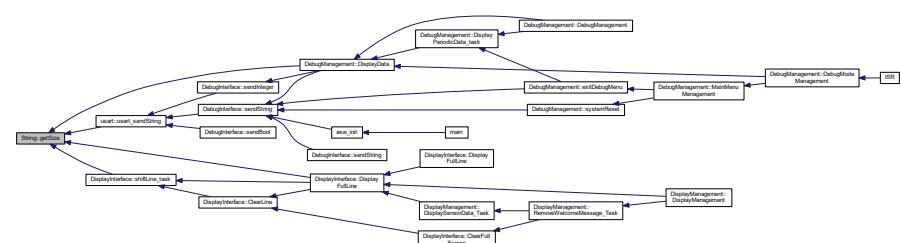
This function returns the size of the string.

#### Returns

Size of the string

Definition at line 64 of file String.h.

Here is the caller graph for this function:



#### 4.17.3.9 `getString()`

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

## String pointer get function.

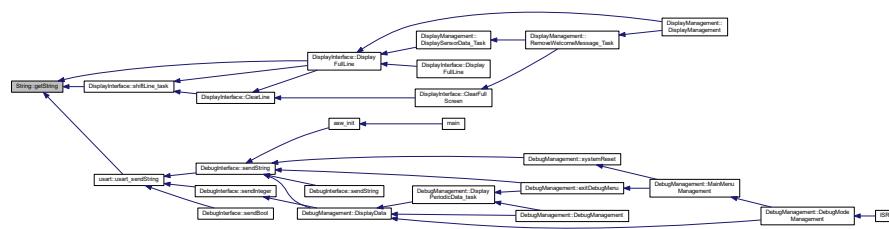
This function returns the pointer to the beginning of the string.

## Returns

## String pointer

Definition at line 53 of file String.h.

Here is the caller graph for this function:



#### 4.17.4 Member Data Documentation

#### 4.17.4.1 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 132 of file String.h.

#### 4.17.4.2 string

`uint8_t* String::string [private]`

Pointer to the start of the string

Definition at line 131 of file String.h.

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

- String.h
  - String.cpp

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

ASW initialization configuration structure.

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

### Public Attributes

- bool `isDebugActivated`
- bool `isLEDActivated`
- bool `isSensorMgtActivated`
- bool `isDisplayActivated`

#### 4.18.1 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.

#### 4.18.2 Member Data Documentation

##### 4.18.2.1 `isDebugActivated`

```
bool T_ASW_init_cnf::isDebugActivated
```

Debug services activation flag

Definition at line 19 of file asw.h.

##### 4.18.2.2 `isDisplayActivated`

```
bool T_ASW_init_cnf::isDisplayActivated
```

LCD display activation flag

Definition at line 22 of file asw.h.

#### 4.18.2.3 isLEDActivated

```
bool T_ASW_init_cnf::isLEDActivated
```

Keep-alive LED activation flag

Definition at line 20 of file asw.h.

#### 4.18.2.4 isSensorMgtActivated

```
bool T_ASW_init_cnf::isSensorMgtActivated
```

Sensor activation

Definition at line 21 of file asw.h.

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

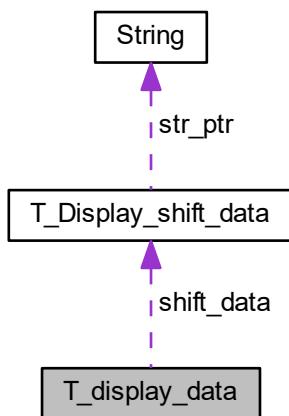
- [asw.h](#)

## 4.19 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]`

### 4.19.1 Detailed Description

Structure containing display data.

This structure contains all data used for screen display

Definition at line 57 of file DisplayInterface.h.

### 4.19.2 Member Data Documentation

#### 4.19.2.1 alignment

`T_DisplayInterface_LineAlignment T_display_data::alignment`

Line alignment

Definition at line 61 of file DisplayInterface.h.

#### 4.19.2.2 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.

#### 4.19.2.3 isEmpty

`bool T_display_data::isEmpty`

Flag indicating if the line is empty or not

Definition at line 59 of file DisplayInterface.h.

#### 4.19.2.4 mode

`T_DisplayInterface_LineDisplayMode T_display_data::mode`

Current display mode

Definition at line 60 of file `DisplayInterface.h`.

#### 4.19.2.5 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:

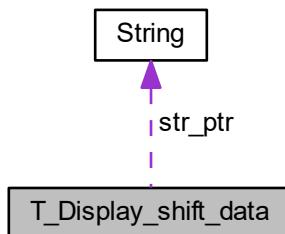
- `DisplayInterface.h`

## 4.20 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`

### 4.20.1 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.

### 4.20.2 Member Data Documentation

#### 4.20.2.1 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.

#### 4.20.2.2 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.

#### 4.20.2.3 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](#)

## 4.21 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`

### 4.21.1 Detailed Description

Structure defining [LCD](#) configuration.

Definition at line 128 of file LCD.h.

### 4.21.2 Member Data Documentation

#### 4.21.2.1 `backlight_en`

```
bool T_LCD_conf_struct::backlight_en
```

Screen backlight enable flag

Definition at line 132 of file LCD.h.

#### 4.21.2.2 `cursor_en`

```
bool T_LCD_conf_struct::cursor_en
```

Screen cursor enable flag

Definition at line 136 of file LCD.h.

#### 4.21.2.3 cursorBlink\_en

```
bool T_LCD_conf_struct::cursorBlink_en
```

Screen cursor blinking enable flag

Definition at line 137 of file LCD.h.

#### 4.21.2.4 display\_en

```
bool T_LCD_conf_struct::display_en
```

Screen display enable flag

Definition at line 135 of file LCD.h.

#### 4.21.2.5 entryModeDir

```
bool T_LCD_conf_struct::entryModeDir
```

Entry mode direction configuration

Definition at line 138 of file LCD.h.

#### 4.21.2.6 entryModeShift

```
bool T_LCD_conf_struct::entryModeShift
```

Entry mode shift configuration

Definition at line 139 of file LCD.h.

#### 4.21.2.7 fontType\_cnf

```
bool T_LCD_conf_struct::fontType_cnf
```

Font configuration

Definition at line 134 of file LCD.h.

#### 4.21.2.8 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.

#### 4.21.2.9 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.

#### 4.21.2.10 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.

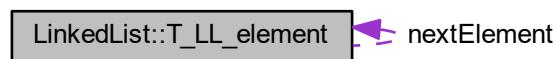
The documentation for this struct was generated from the following file:

- [LCD.h](#)

## 4.22 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](#)

### 4.22.1 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](#).

### 4.22.2 Member Data Documentation

#### 4.22.2.1 data\_ptr

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

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

#### 4.22.2.2 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](#)

## 4.23 T\_SensorManagement\_Sensor\_Config Struct Reference

[Sensor](#) informations structure.

```
#include <sensor_configuration.h>
```

## Public Attributes

- [T\\_SensorManagement\\_Sensor\\_Type](#) [sensor\\_type](#)
- uint16\_t [period](#)
- uint16\_t [validity\\_tmo](#)
- uint8\_t \* [data\\_name\\_str](#)
- uint8\_t \* [unit\\_str](#)

### 4.23.1 Detailed Description

[Sensor](#) informations structure.

This structure contains all configuration informations needed for each used sensor.

Definition at line 17 of file `sensor_configuration.h`.

### 4.23.2 Member Data Documentation

#### 4.23.2.1 `data_name_str`

```
uint8_t* T_SensorManagement_Sensor_Config::data_name_str
```

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

#### 4.23.2.2 `period`

```
uint16_t T_SensorManagement_Sensor_Config::period
```

Definition at line 20 of file `sensor_configuration.h`.

#### 4.23.2.3 `sensor_type`

```
T_SensorManagement_Sensor_Type T_SensorManagement_Sensor_Config::sensor_type
```

Definition at line 19 of file `sensor_configuration.h`.

#### 4.23.2.4 `unit_str`

```
uint8_t* T_SensorManagement_Sensor_Config::unit_str
```

Definition at line 23 of file `sensor_configuration.h`.

### 4.23.2.5 validity\_tmo

```
uint16_t T_SensorManagement_Sensor_Config::validity_tmo
```

Definition at line 21 of file sensor\_configuration.h.

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

- [sensor\\_configuration.h](#)

## 4.24 scheduler::Task\_t Struct Reference

Type defining a task structure.

### Public Attributes

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

### 4.24.1 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.

### 4.24.2 Member Data Documentation

#### 4.24.2.1 period

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

Period of the task

Definition at line 132 of file scheduler.h.

#### 4.24.2.2 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:

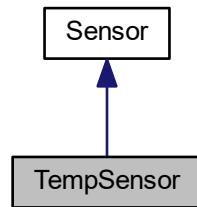
- [scheduler.h](#)

## 4.25 TempSensor Class Reference

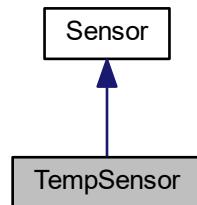
Class for temperature sensor.

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

Inheritance diagram for TempSensor:



Collaboration diagram for TempSensor:



## Public Member Functions

- [TempSensor \(\)](#)  
*Class constructor.*
- [TempSensor \(uint16\\_t val\\_tmo, uint16\\_t period\)](#)  
*Overloaded class constructor.*
- [bool updateTaskPeriod \(uint16\\_t period\)](#)  
*Task period update.*

## Static Public Member Functions

- [static void readTempSensor\\_task \(\)](#)  
*Task for reading temperature values.*

## Additional Inherited Members

### 4.25.1 Detailed Description

Class for temperature sensor.

This class defines all functions used to read data from temperature sensor and monitor it. It is inherited from class [Sensor](#).

Definition at line 20 of file [TempSensor.h](#).

### 4.25.2 Constructor & Destructor Documentation

#### 4.25.2.1 TempSensor() [1/2]

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

Class constructor.

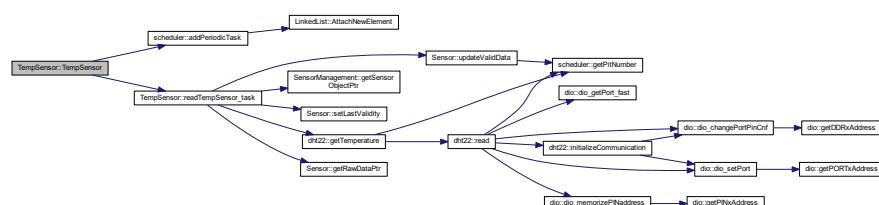
This function initializes all data of the class [TempSensor](#). If needed, it creates a new instance of the DHT22 sensor object. It also adds the periodic task in the scheduler.

#### Returns

Nothing

Definition at line 26 of file [TempSensor.cpp](#).

Here is the call graph for this function:



#### 4.25.2.2 TempSensor() [2/2]

```
TempSensor::TempSensor (
    uint16_t val_tmo,
    uint16_t period )
```

Overloaded class constructor.

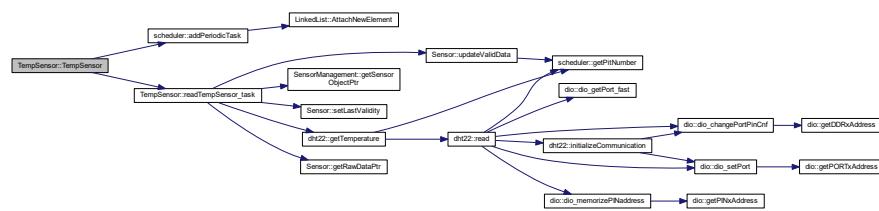
This function initializes all data of the class [TempSensor](#). It sets validity timeout and task period to the given value. If needed, it creates a new instance of the DHT22 sensor object. It also adds the periodic task in the scheduler.

#### Returns

Nothing

Definition at line 36 of file [TempSensor.cpp](#).

Here is the call graph for this function:



### 4.25.3 Member Function Documentation

#### 4.25.3.1 readTempSensor\_task()

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

Task for reading temperature values.

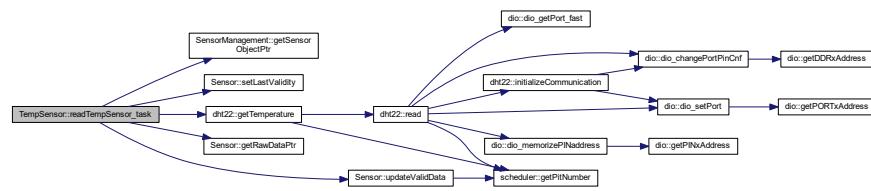
This task reads temperature data using DHT22 driver. It is called periodically.

**Returns**

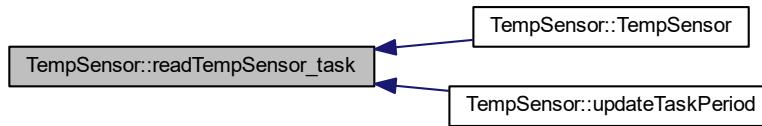
Nothing

Definition at line 46 of file TempSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.25.3.2 updateTaskPeriod()**

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

Task period update.

This function updates the period of the temperature task.

**Parameters**

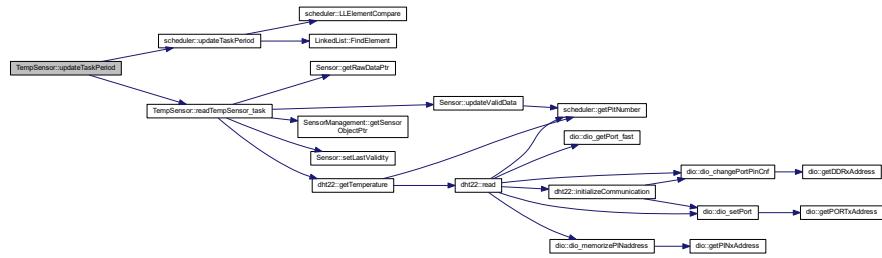
in	<i>period</i>	New period of the task
----	---------------	------------------------

**Returns**

True if the period has been updated, false otherwise

Definition at line 60 of file TempSensor.cpp.

Here is the call graph for this function:



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

- TempSensor.h
  - TempSensor.cpp

## 4.26 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`

#### 4.26.1 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.

## 4.26.2 Constructor & Destructor Documentation

### 4.26.2.1 timer()

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

Class constructor.

This function initializes class attributes

#### Returns

Nothing

Definition at line 15 of file timer.cpp.

## 4.26.3 Member Function Documentation

### 4.26.3.1 configureTimer1()

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

Configures Timer #1.

This function configures hardware timer #1 in CTC mode, enables its interrupts, sets prescaler to *a\_prescaler* and CTC value to *a\_ctcValue*

#### Parameters

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

#### Returns

Nothing

Definition at line 20 of file timer.cpp.

Here is the caller graph for this function:



#### 4.26.3.2 getTimer1Value()

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

Reads current value of timer #1.

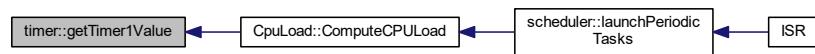
This function reads the value of of timer #1 using register TCNT1. The function is inlined to speed up SW execution.

##### Returns

Current timer value

Definition at line 58 of file timer.h.

Here is the caller graph for this function:



#### 4.26.3.3 startTimer1()

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

Start Timer #1.

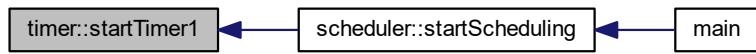
This functions starts Timer #1. Timer shall be initialized before this function is called.

**Returns**

Nothing

Definition at line 58 of file timer.cpp.

Here is the caller graph for this function:



#### 4.26.3.4 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.

#### 4.26.4 Member Data Documentation

##### 4.26.4.1 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](#)

## 4.27 usart Class Reference

USART serial bus class.

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

### Public Member Functions

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

### Private Member Functions

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

### Private Attributes

- uint16\_t **BaudRate**

#### 4.27.1 Detailed Description

USART serial bus class.

This class defines all useful functions for USART serial bus

Definition at line 16 of file usart.h.

#### 4.27.2 Constructor & Destructor Documentation

##### 4.27.2.1 usart()

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

Class usart constructor.

Initializes the class and call hardware initialization function

**Parameters**

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

**Returns**

Nothing.

Definition at line 18 of file usart.cpp.

Here is the call graph for this function:



### 4.27.3 Member Function Documentation

#### 4.27.3.1 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.

#### 4.27.3.2 usart\_init()

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

USART hardware initialization.

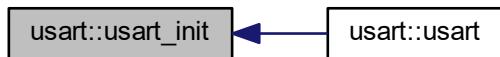
This function will initialize the USART using selected baudrate. User must pay attention to select one of the usually used Baud Rate (9600, 19200, 38400, 57600). Note that since an uint16 is used as argument, Baud rate cannot be more than 57600.

##### Returns

Nothing.

Definition at line 25 of file usart.cpp.

Here is the caller graph for this function:



#### 4.27.3.3 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.

#### 4.27.3.4 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:

**4.27.3.5 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**

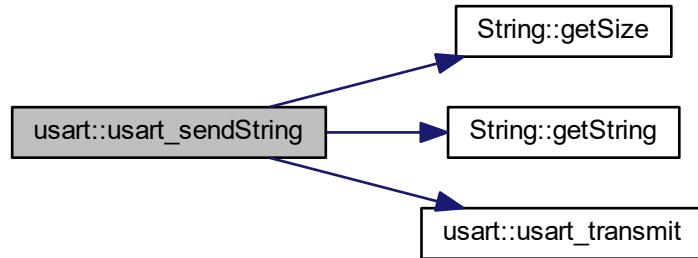
in	<i>str</i>	Pointer to the string being sent
----	------------	----------------------------------

**Returns**

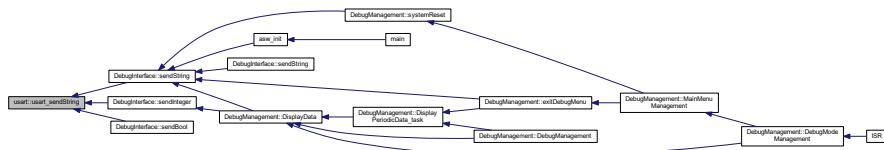
Nothing.

Definition at line 48 of file usart.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.27.3.6 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

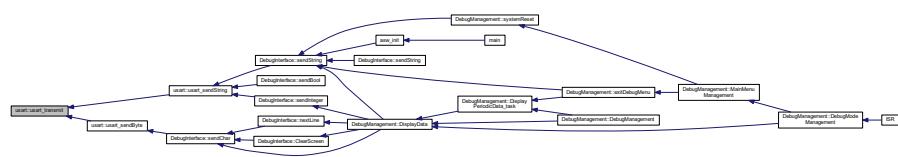
in	<i>Data</i>	Desired data char to transmit
----	-------------	-------------------------------

##### Returns

Nothing.

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

Here is the caller graph for this function:



#### 4.27.4 Member Data Documentation

#### 4.27.4.1 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`

## 4.28 Watchdog Class Reference

## Watchdog management class.

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

## Public Member Functions

- **Watchdog ()**  
*Class constructor.*
  - **Watchdog (uint8\_t timeout)**  
*Overloaded class constructor.*
  - **void reset ()**  
*Watchdog reset function.*
  - **void timeoutUpdate (uint8\_t value)**  
*Watchdog timeout value update function.*
  - **void SystemReset ()**  
*System reset function.*
  - **uint16\_t getTMOValue ()**  
*Watchdog timeout get value.*
  - **bool isEnabled ()**  
*Watchdog status function.*
  - **bool SwitchWdg ()**  
*Watchdog switching function.*

## Private Member Functions

- void **disable** ()
 

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

*Watchdog enabling function.*

## Private Attributes

- uint8\_t **tmo\_value**
- bool **isActive**

### 4.28.1 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.

### 4.28.2 Constructor & Destructor Documentation

#### 4.28.2.1 Watchdog() [1/2]

Watchdog::Watchdog ( )

Class constructor.

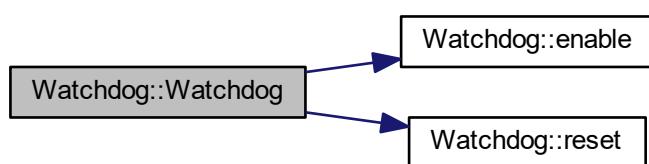
This function initializes the watchdog class. It enables the HW watchdog with a default timeout value.

#### Returns

Nothing

Definition at line 19 of file Watchdog.cpp.

Here is the call graph for this function:



### 4.28.2.2 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

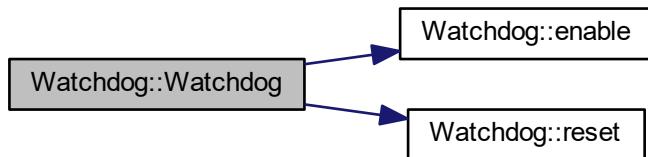
in	<i>timeout</i>	Timeout value requested for the watchdog
----	----------------	--

#### Returns

Nothing

Definition at line 26 of file Watchdog.cpp.

Here is the call graph for this function:



### 4.28.3 Member Function Documentation

#### 4.28.3.1 disable()

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

[Watchdog](#) disabling function.

This function disables the watchdog by calling wdt\_disable macro.

**Returns**

Nothing

Definition at line 44 of file Watchdog.cpp.

Here is the caller graph for this function:

**4.28.3.2 enable()**

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

[Watchdog](#) enabling function.

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

**Parameters**

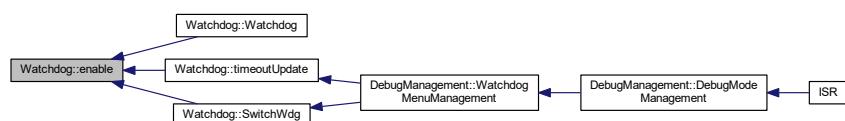
in	value	Timeout value
----	-------	---------------

**Returns**

Nothing

Definition at line 34 of file Watchdog.cpp.

Here is the caller graph for this function:



#### 4.28.3.3 getTMOValue()

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

[Watchdog](#) timeout get value.

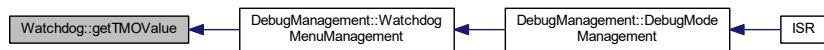
This function returns the current watchdog timeout value in ms. It has to convert the value of tmo\_value into a numeric value of the timeout.

##### Returns

Timeout value.

Definition at line 75 of file Watchdog.cpp.

Here is the caller graph for this function:



#### 4.28.3.4 isEnabled()

```
bool Watchdog::isEnabled ( ) [inline]
```

[Watchdog](#) status function.

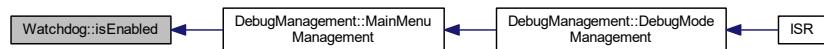
This function returns the current status of the watchdog : enabled or disabled.

##### Returns

True if the watchdog is enabled, false otherwise.

Definition at line 91 of file Watchdog.h.

Here is the caller graph for this function:



#### 4.28.3.5 reset()

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

[Watchdog](#) reset function.

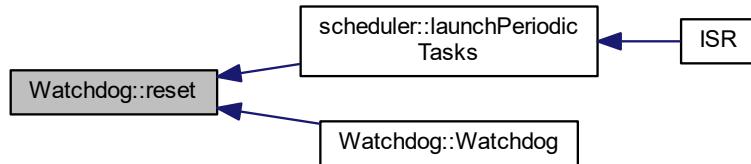
This function resets the watchdog timer by calling wdt\_reset macro

##### Returns

Nothing

Definition at line 53 of file Watchdog.cpp.

Here is the caller graph for this function:



#### 4.28.3.6 SwitchWdg()

```
bool Watchdog::SwitchWdg ( )
```

[Watchdog](#) switching function.

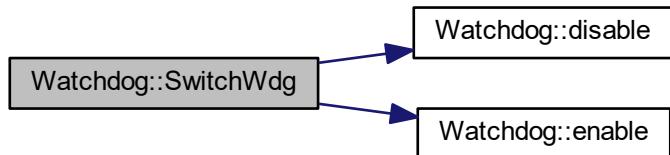
This function switches the state of the watchdog. If it was enabled, the function disables the watchdog, and if it was disabled, the function enables it with the memorized timeout value. The function returns the new status of the watchdog.

**Returns**

New status of the watchdog : True if enabled, false if disabled.

Definition at line 117 of file Watchdog.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.28.3.7 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 70 of file Watchdog.cpp.

Here is the caller graph for this function:



#### 4.28.3.8 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

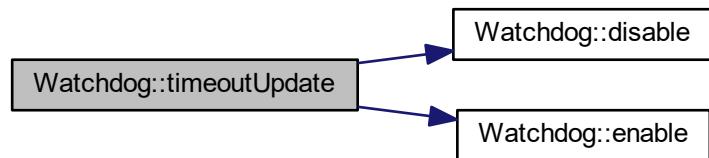
in	value	New timeout value
----	-------	-------------------

##### Returns

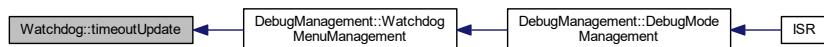
Nothing

Definition at line 58 of file Watchdog.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.28.4 Member Data Documentation

##### 4.28.4.1 isActive

```
bool Watchdog::isActive [private]
```

[Watchdog](#) activation flag

Definition at line 109 of file Watchdog.h.

#### 4.28.4.2 tmo\_value

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

Current timeout value

Definition at line 108 of file Watchdog.h.

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

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



# Chapter 5

# File Documentation

## 5.1 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 "sensors_mgt/SensorManagement.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:

Include dependency graph for asw.cpp:



## Functions

- void asw\_init ()

## *Initialization of ASW.*

### 5.1.1 Detailed Description

ASW main file.

#### Date

15 mars 2018

#### Author

nicls67

### 5.1.2 Function Documentation

#### 5.1.2.1 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.

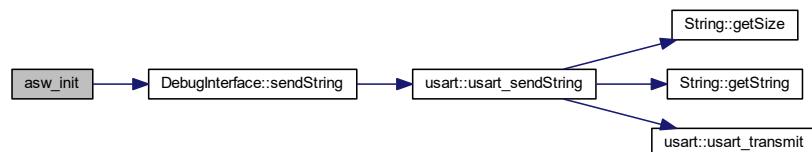
This function shall be called after BSW initialization function.

#### Returns

Nothing

Definition at line 35 of file asw.cpp.

Here is the call graph for this function:



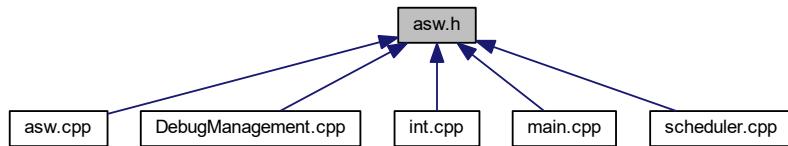
Here is the caller graph for this function:



## 5.2 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.*

#### 5.2.1 Detailed Description

ASW main header file.

##### Date

15 mars 2018

##### Author

nicls67

#### 5.2.2 Function Documentation

### 5.2.2.1 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.

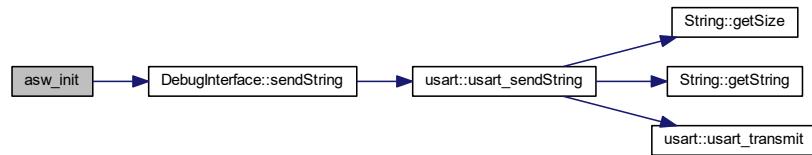
This function shall be called after BSW initialization function.

#### Returns

Nothing

Definition at line 35 of file asw.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



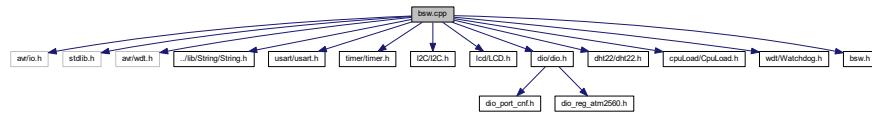
## 5.3 bsw.cpp File Reference

BSW main file.

```
#include <avr/io.h>
#include <stdlib.h>
#include <avr/wdt.h>
#include "../lib/String/String.h"
#include "uart/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.*

### 5.3.1 Detailed Description

BSW main file.

Date

13 mars 2018

Author

nicls67

### 5.3.2 Function Documentation

#### 5.3.2.1 bsw\_init()

void bsw\_init ( )

*Initialization of BSW.*

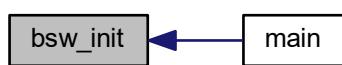
This function instantiates all driver objects, leading hardware initialization. The addresses of driver objects are then stored in BSW\_cnf\_struct structure.

Returns

Nothing

Definition at line 26 of file bsw.cpp.

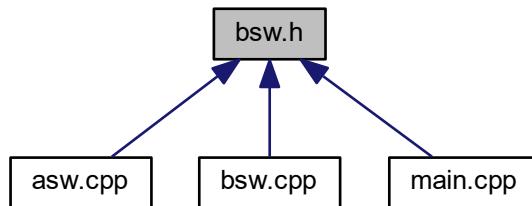
Here is the caller graph for this function:



## 5.4 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.*

### 5.4.1 Detailed Description

BSW main header file.

#### Date

13 mars 2018

#### Author

nicls67

### 5.4.2 Function Documentation

#### 5.4.2.1 bsw\_init()

```
void bsw_init ( )
```

Initialization of BSW.

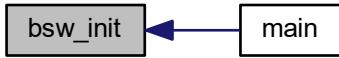
This function instantiates all driver objects, leading hardware initialization. The addresses of driver objects are then stored in BSW\_cnf\_struct structure.

#### Returns

Nothing

Definition at line 26 of file bsw.cpp.

Here is the caller graph for this function:

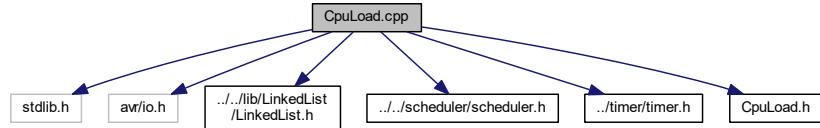


## 5.5 CpuLoad.cpp File Reference

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](#)

### 5.5.1 Detailed Description

Defines functions of class [CpuLoad](#).

#### Date

21 mars 2019

#### Author

nicls67

### 5.5.2 Variable Documentation

#### 5.5.2.1 p\_global\_BSW\_cpuload

`CpuLoad* p_global_BSW_cpuload`

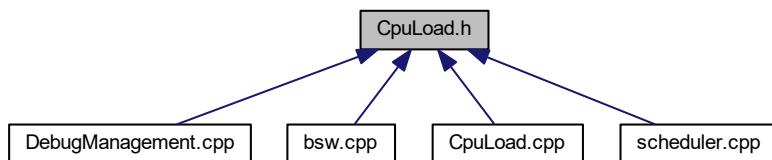
Pointer to cpu load library object

Definition at line 18 of file CpuLoad.cpp.

## 5.6 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`

### 5.6.1 Detailed Description

`CpuLoad` class header file.

#### Date

21 mars 2019

#### Author

nicls67

### 5.6.2 Macro Definition Documentation

#### 5.6.2.1 NB\_OF\_SAMPLES

```
#define NB_OF_SAMPLES 50
```

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

### 5.6.3 Variable Documentation

#### 5.6.3.1 p\_global\_BSW\_cpuload

`CpuLoad* p_global_BSW_cpuload`

Pointer to cpu load library object

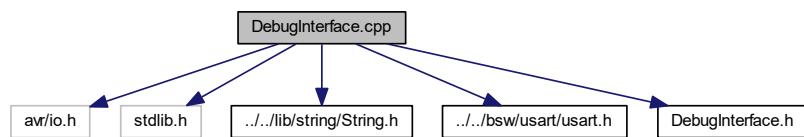
Definition at line 18 of file `CpuLoad.cpp`.

## 5.7 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`

#### 5.7.1 Detailed Description

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

##### Date

15 mars 2018

##### Author

nicls67

#### 5.7.2 Variable Documentation

##### 5.7.2.1 `p_global_ASW_DebugInterface`

`DebugInterface* p_global_ASW_DebugInterface`

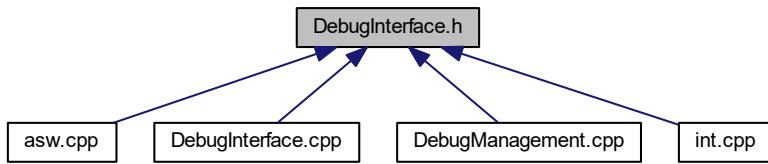
Pointer to USART debug interface object

Definition at line 19 of file DebugInterface.cpp.

## 5.8 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](#)

#### 5.8.1 Detailed Description

Header file for debug and logging functions.

##### Date

15 mars 2018

##### Author

nicls67

#### 5.8.2 Macro Definition Documentation

### 5.8.2.1 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.

### 5.8.3 Variable Documentation

#### 5.8.3.1 p\_global\_ASW\_DebugInterface

```
DebugInterface* p_global_ASW_DebugInterface
```

Pointer to USART debug interface object

Definition at line 19 of file DebugInterface.cpp.

## 5.9 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 "../../sensors/Sensor.h"
#include "../../sensors/TempSensor/TempSensor.h"
#include "../../sensors/HumSensor/HumSensor.h"
#include "../../sensors_mgt/SensorManagement.h"
#include "../debug_ift/DebugInterface.h"
#include "DebugManagement.h"
#include "../asw.h"
#include "../../main.h"
```

Include dependency graph for DebugManagement.cpp:



## Variables

- `DebugManagement * p_global_ASW_DebugManagement`  
`const uint8_t str_debug_main_menu []`  
*Main menu of debug mode.*
- `const uint8_t str_debug_wdg_menu []`  
*Watchdog menu of debug mode.*
- `const uint8_t str_debug_wdg_timeout_update_selection []`  
*Watchdog timeout update selection.*
- `const uint8_t str_debug_info_message_wrong_menu_selection [] = "Impossible de faire ca... !"`  
*Info menu string in case a wrong selection has been performed.*
- `const uint8_t str_debug_info_message_wdg_tmo_updated [] = "Valeur modifiee !"`  
*Info menu string in case the watchdog timeout value has been updated.*
- `const uint8_t str_debug_info_message_wdg_tmo_value [] = "Valeur du timeout watchdog (ms) : "`  
*Info menu string displaying the current value of the watchdog timeout.*
- `const uint8_t str_debug_info_message_wdg_disabled [] = "Watchdog inactif !"`  
*Info menu string displayed when the watchdog has been disabled.*
- `const uint8_t str_debug_info_message_wdg_enabled [] = "Watchdog actif !"`  
*Info menu string displayed when the watchdog has been enabled.*

### 5.9.1 Detailed Description

Debug management class source file.

#### Date

8 mai 2019

#### Author

nicls67

### 5.9.2 Variable Documentation

#### 5.9.2.1 p\_global\_ASW\_DebugManagement

`DebugManagement * p_global_ASW_DebugManagement`

Pointer to the `DebugManagement` object

Definition at line 33 of file `DebugManagement.cpp`.

### 5.9.2.2 str\_debug\_info\_message\_wdg\_disabled

```
const uint8_t str_debug_info_message_wdg_disabled[ ] = "Watchdog inactif !"
```

Info menu string displayed when the watchdog has been disabled.

Definition at line 94 of file DebugManagement.cpp.

### 5.9.2.3 str\_debug\_info\_message\_wdg\_enabled

```
const uint8_t str_debug_info_message_wdg_enabled[ ] = "Watchdog actif !"
```

Info menu string displayed when the watchdog has been enabled.

Definition at line 99 of file DebugManagement.cpp.

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

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

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

Definition at line 84 of file DebugManagement.cpp.

### 5.9.2.5 str\_debug\_info\_message\_wdg\_tmo\_value

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

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

Definition at line 89 of file DebugManagement.cpp.

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

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

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

Definition at line 79 of file DebugManagement.cpp.

### 5.9.2.7 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 40 of file DebugManagement.cpp.

### 5.9.2.8 str\_debug\_wdg\_menu

```
const uint8_t str_debug_wdg_menu[ ]
```

**Initial value:**

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

Watchdog menu of debug mode.

Definition at line 50 of file DebugManagement.cpp.

### 5.9.2.9 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"
```

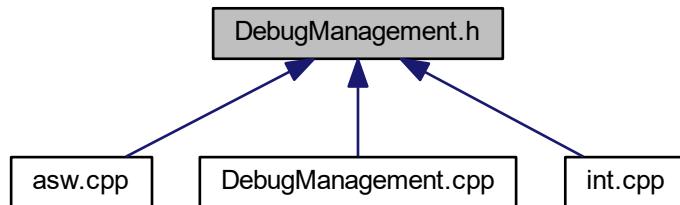
Watchdog timeout update selection.

Definition at line 61 of file DebugManagement.cpp.

## 5.10 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`

#### 5.10.1 Detailed Description

Debug management class header file.

##### Date

8 mai 2019

##### Author

nicls67

## 5.10.2 Macro Definition Documentation

### 5.10.2.1 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.

### 5.10.2.2 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.

## 5.10.3 Enumeration Type Documentation

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

```
enum debug_mgt_main_menu_state_t
```

Defines the debug states.

Enumerator

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

Definition at line 20 of file DebugManagement.h.

### 5.10.3.2 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.

#### 5.10.4 Variable Documentation

##### 5.10.4.1 p\_global\_ASW\_DebugManagement

`DebugManagement* p_global_ASW_DebugManagement`

Pointer to the `DebugManagement` object

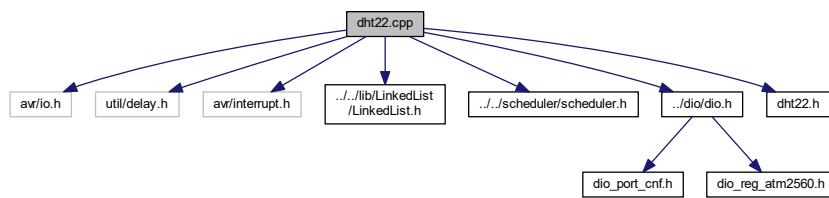
Definition at line 33 of file DebugManagement.cpp.

## 5.11 dht22.cpp File Reference

This file defines classes for DHT22 driver.

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "../lib/LinkedList/LinkedList.h"
#include "../scheduler/scheduler.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`

### 5.11.1 Detailed Description

This file defines classes for DHT22 driver.

#### Date

23 mars 2018

#### Author

nicls67

### 5.11.2 Macro Definition Documentation

#### 5.11.2.1 MAX\_WAIT\_TIME\_US

```
#define MAX_WAIT_TIME_US 100
```

Maximum waiting time in microseconds

Definition at line 20 of file dht22.cpp.

### 5.11.3 Variable Documentation

#### 5.11.3.1 p\_global\_BSW\_dht22

```
dht22* p_global_BSW_dht22
```

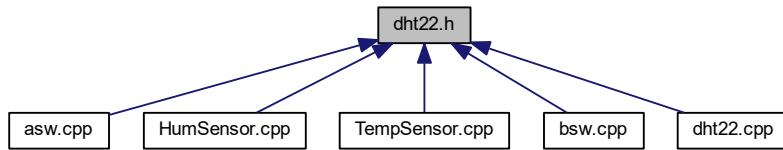
Pointer to `dht22` driver object

Definition at line 22 of file dht22.cpp.

## 5.12 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](#)

#### 5.12.1 Detailed Description

DHT22 driver header file.

##### Date

23 mars 2018

##### Author

nicls67

#### 5.12.2 Variable Documentation

##### 5.12.2.1 [p\\_global\\_BSW\\_dht22](#)

[dht22\\*](#) [p\\_global\\_BSW\\_dht22](#)

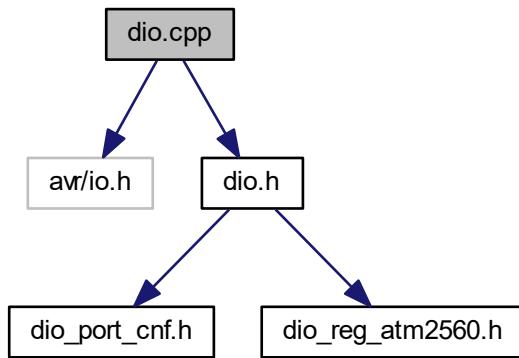
Pointer to [dht22](#) driver object

Definition at line 22 of file [dht22.cpp](#).

## 5.13 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](#)

#### 5.13.1 Detailed Description

DIO library.

##### Date

13 mars 2018

##### Author

nicls67

#### 5.13.2 Variable Documentation

### 5.13.2.1 p\_global\_BSW\_dio

`dio* p_global_BSW_dio`

Pointer to dio driver object

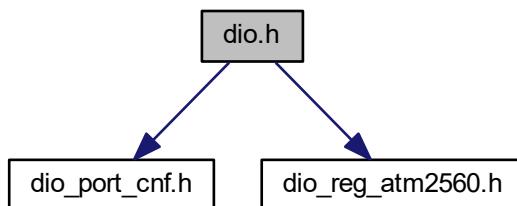
Definition at line 14 of file dio.cpp.

## 5.14 dio.h File Reference

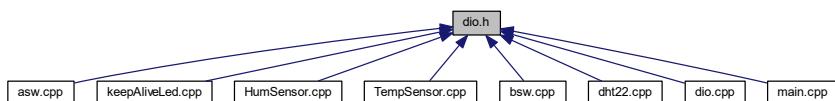
DIO library header file.

```
#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`

### 5.14.1 Detailed Description

DIO library header file.

#### Date

13 mars 2018

#### Author

nicls67

### 5.14.2 Macro Definition Documentation

#### 5.14.2.1 DECODE\_PIN

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

Macro used to extract pin index

Definition at line 20 of file dio.h.

#### 5.14.2.2 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.

#### 5.14.2.3 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.

#### 5.14.2.4 PORT\_CNF\_IN

```
#define PORT_CNF_IN 0
```

Pin is configured as input

Definition at line 16 of file dio.h.

#### 5.14.2.5 PORT\_CNF\_OUT

```
#define PORT_CNF_OUT 1
```

Pin is configured as output

Definition at line 15 of file dio.h.

### 5.14.3 Variable Documentation

#### 5.14.3.1 p\_global\_BSW\_dio

```
dio* p_global_BSW_dio
```

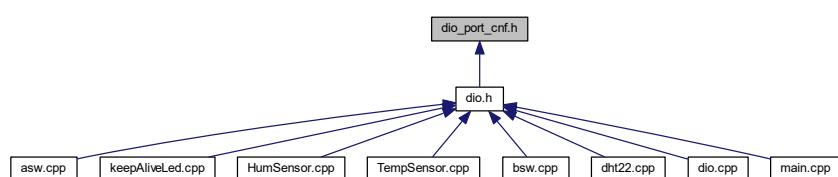
Pointer to dio driver object

Definition at line 14 of file dio.cpp.

## 5.15 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`

### 5.15.1 Detailed Description

Digital ports configuration file.

#### Date

19 mars 2019

#### Author

nicls67

### 5.15.2 Macro Definition Documentation

#### 5.15.2.1 PORT\_A

`#define PORT_A 0`

PORTA index

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

#### 5.15.2.2 PORT\_B

`#define PORT_B 1`

PORPB index

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

### 5.15.2.3 PORT\_C

```
#define PORT_C 2
```

PORTC index

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

### 5.15.2.4 PORT\_D

```
#define PORT_D 3
```

PORTD index

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

### 5.15.2.5 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.

### 5.15.2.6 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

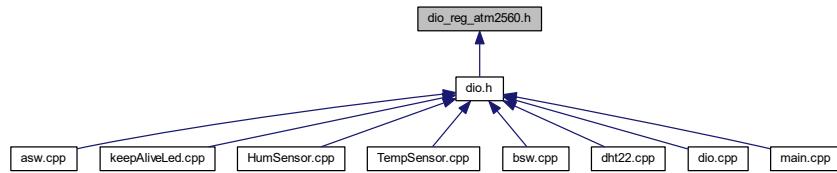
PB7 : LOW

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

## 5.16 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)

### 5.16.1 Detailed Description

Defines DIO register addresses for ATMEGA2560.

#### Date

19 mars 2019

#### Author

nicls67

### 5.16.2 Macro Definition Documentation

### 5.16.2.1 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.

### 5.16.2.2 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.

### 5.16.2.3 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.

### 5.16.2.4 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.

### 5.16.2.5 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.

### 5.16.2.6 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.

### 5.16.2.7 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.

### 5.16.2.8 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.

### 5.16.2.9 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.

### 5.16.2.10 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.

### 5.16.2.11 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.

### 5.16.2.12 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.

## 5.17 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](#)

### 5.17.1 Detailed Description

Source code file for display services.

#### Date

23 avr. 2019

#### Author

nicls67

### 5.17.2 Variable Documentation

#### 5.17.2.1 p\_global\_ASW\_DisplayInterface

`DisplayInterface* p_global_ASW_DisplayInterface`

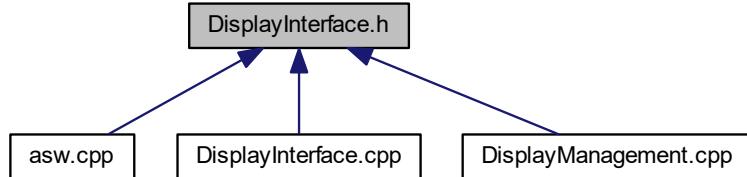
Pointer to `DisplayInterface` object

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

## 5.18 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`

### 5.18.1 Detailed Description

`DisplayInterface` class header file.

Date

23 avr. 2019

Author

nicls67

### 5.18.2 Macro Definition Documentation

#### 5.18.2.1 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`.

#### 5.18.2.2 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`.

### 5.18.3 Enumeration Type Documentation

#### 5.18.3.1 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.

**5.18.3.2 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.

**5.18.4 Variable Documentation****5.18.4.1 p\_global\_ASW\_DisplayInterface**

```
DisplayInterface* p_global_ASW_DisplayInterface
```

Pointer to [DisplayInterface](#) object

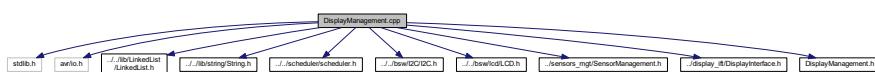
Definition at line 25 of file DisplayInterface.cpp.

## 5.19 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 "../sensors_mgt/SensorManagement.h"
#include "../display_ift/DisplayInterface.h"
#include "DisplayManagement.h"
```

Include dependency graph for DisplayManagement.cpp:



### Variables

- `DisplayManagement * p_global_ASW_DisplayManagement`
- `const uint8_t welcomeMessageString [] = "Bienvenue !"`
- `const uint8_t noSensorsDisplayString [] = "Capteurs desactives"`

### 5.19.1 Detailed Description

Display management source file.

#### Date

1 mai 2019

#### Author

nicls67

### 5.19.2 Variable Documentation

#### 5.19.2.1 noSensorsDisplayString

```
const uint8_t noSensorsDisplayString[] = "Capteurs desactives"
```

**String** used in case sensors are deactivated

Definition at line 28 of file DisplayManagement.cpp.

### 5.19.2.2 p\_global\_ASW\_DisplayManagement

`DisplayManagement* p_global_ASW_DisplayManagement`

Pointer to `DisplayManagement` object

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

### 5.19.2.3 welcomeMessageString

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

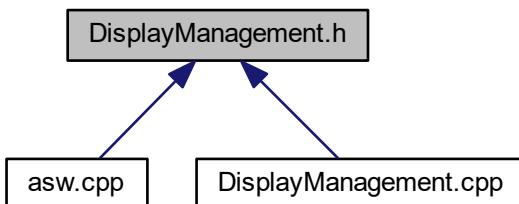
`String` displayed on the screen at startup

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

## 5.20 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_FIRST_LINE_SENSORS 0`
- `#define DISPLAY_MGT_I2C_BITRATE (uint32_t)100000`

## Variables

- const `T_LCD_conf_struct LCD_init_cnf`  
*LCD configuration structure.*
- `DisplayManagement * p_global_ASW_DisplayManagement`

### 5.20.1 Detailed Description

Display management class header file.

#### Date

1 mai 2019

#### Author

nicls67

### 5.20.2 Macro Definition Documentation

#### 5.20.2.1 DISPLAY\_MGT\_FIRST\_LINE\_SENSORS

```
#define DISPLAY_MGT_FIRST_LINE_SENSORS 0
```

Sensors data are displayed starting on line 0

Definition at line 18 of file DisplayManagement.h.

#### 5.20.2.2 DISPLAY\_MGT\_I2C\_BITRATE

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

I2C bus bitrate is 100 kHz

Definition at line 20 of file DisplayManagement.h.

#### 5.20.2.3 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.

#### 5.20.2.4 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.

#### 5.20.2.5 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.

### 5.20.3 Variable Documentation

#### 5.20.3.1 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 26 of file DisplayManagement.h.

#### 5.20.3.2 p\_global\_ASW\_DisplayManagement

```
DisplayManagement* p_global_ASW_DisplayManagement
```

Pointer to [DisplayManagement](#) object

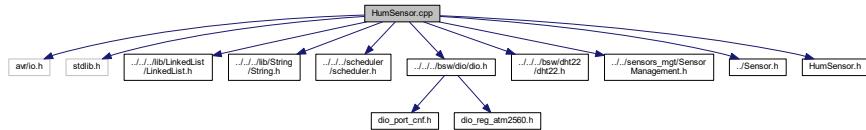
Definition at line 25 of file DisplayManagement.cpp.

## 5.21 HumSensor.cpp File Reference

Defines function of class [HumSensor](#).

```
#include <avr/io.h>
#include <stdlib.h>
#include "../../../lib/LinkedList/LinkedList.h"
#include "../../../lib/String/String.h"
#include "../../../scheduler/scheduler.h"
#include "../../../bsw/dio/dio.h"
#include "../../../bsw/dht22/dht22.h"
#include "../../../sensors_mgt/SensorManagement.h"
#include "../Sensor.h"
#include "HumSensor.h"
```

Include dependency graph for HumSensor.cpp:



### Macros

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

#### 5.21.1 Detailed Description

Defines function of class [HumSensor](#).

##### Date

20 juin 2019

##### Author

nicls67

#### 5.21.2 Macro Definition Documentation

##### 5.21.2.1 DHT22\_PORT

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

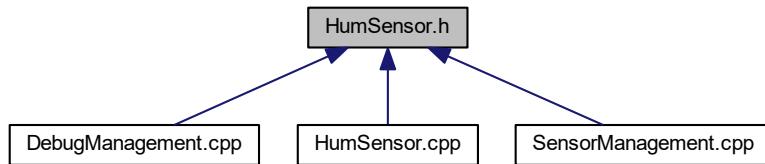
DHT22 is connected to port PB6

Definition at line 26 of file [HumSensor.cpp](#).

## 5.22 HumSensor.h File Reference

Class [HumSensor](#) header file.

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



### Classes

- class [HumSensor](#)  
*Class for humidity sensor.*

#### 5.22.1 Detailed Description

Class [HumSensor](#) header file.

Date

20 juin 2019

Author

nicls67

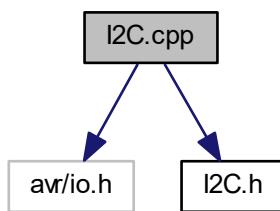
## 5.23 I2C.cpp File Reference

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`

### 5.23.1 Detailed Description

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

#### Date

19 avr. 2019

#### Author

nicls67

### 5.23.2 Variable Documentation

#### 5.23.2.1 `p_global_BSW_i2c`

`I2C* p_global_BSW_i2c`

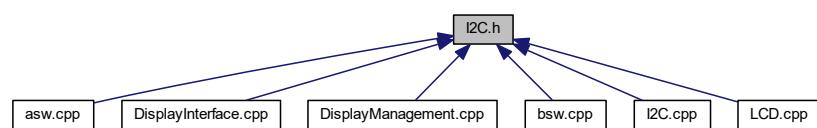
Pointer to [I2C](#) driver object

Definition at line 14 of file I2C.cpp.

## 5.24 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

### 5.24.1 Detailed Description

I2C class header file.

#### Date

19 avr. 2019

#### Author

nicls67

### 5.24.2 Macro Definition Documentation

#### 5.24.2.1 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.

#### 5.24.2.2 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.

### 5.24.2.3 START

```
#define START 0x08
```

TWSR status code : START condition transmitted

Definition at line 13 of file I2C.h.

### 5.24.3 Variable Documentation

#### 5.24.3.1 p\_global\_BSW\_i2c

```
I2C* p_global_BSW_i2c
```

Pointer to [I2C](#) driver object

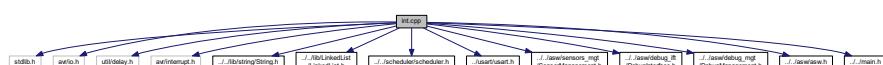
Definition at line 14 of file I2C.cpp.

## 5.25 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/sensors_mgt/SensorManagement.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.*

### 5.25.1 Detailed Description

Interrupt management source file.

#### Date

22 mai 2019

#### Author

nicls67

### 5.25.2 Function Documentation

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

```
ISR (
    TIMER1_COMPA_vect )
```

Main software interrupt.

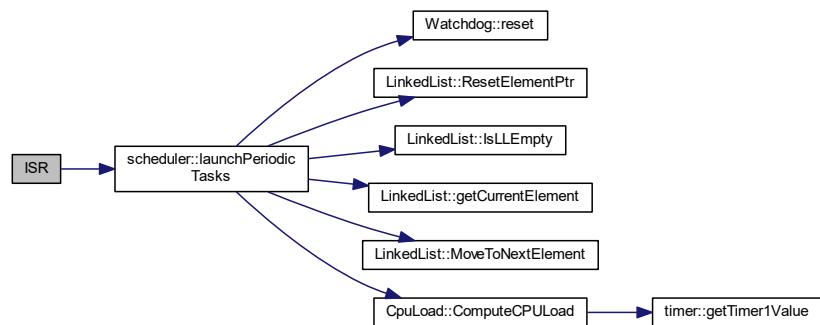
This function handles the interrupt raised by Timer #1. It wakes up the software every 500 ms to perform applications.

#### Returns

Nothing

Definition at line 34 of file int.cpp.

Here is the call graph for this function:



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

```
ISR ( USART0_RX_vect )
```

USART Rx Complete interrupt.

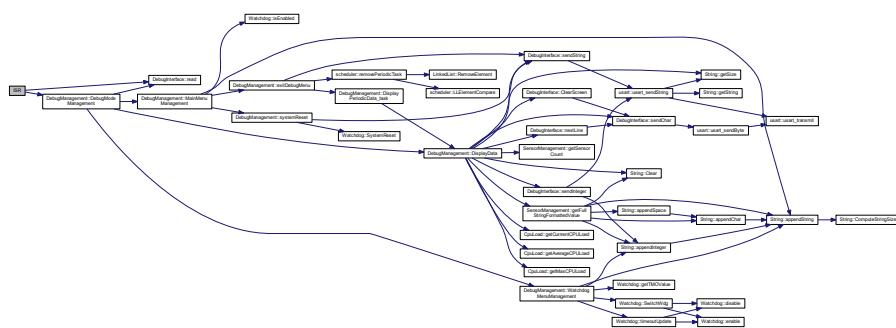
This function handles the interrupt raised when a frame has been received by USART. If debug mode mode is active, it calls debug mode management function. If inactive, it calls debug mode activation function if the received character is 'a'

## Returns

Nothing

Definition at line 46 of file int.cpp.

Here is the call graph for this function:



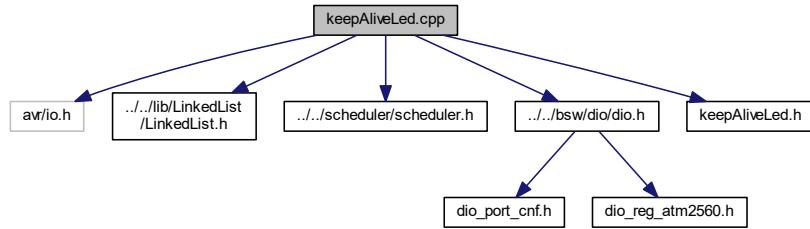
## 5.26 keepAliveLed.cpp File Reference

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:

Include dependency graph for keepAliveLed.cpp:



## Variables

- `keepAliveLed * p_global_ASW_keepAliveLed`

### 5.26.1 Detailed Description

Definition of function for class `keepAliveLed`.

#### Date

17 mars 2018

#### Author

nicls67

### 5.26.2 Variable Documentation

#### 5.26.2.1 `p_global_ASW_keepAliveLed`

`keepAliveLed* p_global_ASW_keepAliveLed`

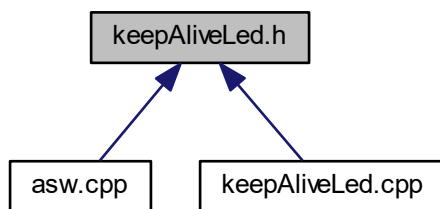
Pointer to `keepAliveLed` object

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

## 5.27 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](#)

### 5.27.1 Detailed Description

Class [keepAliveLed](#) header file.

#### Date

17 mars 2018

#### Author

nicls67

### 5.27.2 Macro Definition Documentation

#### 5.27.2.1 LED\_PORT

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

LED is connected to port PB7

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

#### 5.27.2.2 PERIOD\_MS\_TASK\_LED

```
#define PERIOD_MS_TASK_LED SW_PERIOD_MS
```

Period for led blinking

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

### 5.27.3 Variable Documentation

#### 5.27.3.1 p\_global\_ASW\_keepAliveLed

`keepAliveLed* p_global_ASW_keepAliveLed`

Pointer to `keepAliveLed` object

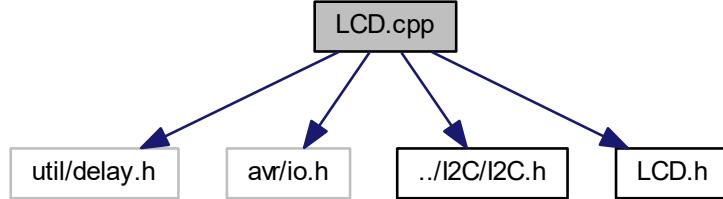
Definition at line 20 of file `keepAliveLed.cpp`.

## 5.28 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`

### 5.28.1 Detailed Description

[LCD](#) class source file.

#### Date

20 avr. 2019

#### Author

nicls67

## 5.28.2 Variable Documentation

### 5.28.2.1 p\_global\_BSW\_lcd

`LCD* p_global_BSW_lcd`

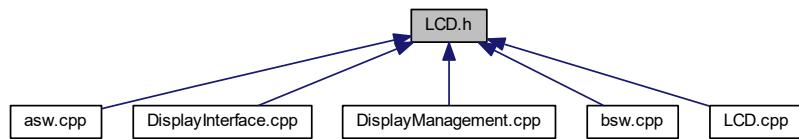
Pointer to `LCD` driver object

Definition at line 16 of file LCD.cpp.

## 5.29 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

### 5.29.1 Detailed Description

LCD class header file.

Date

20 avr. 2019

Author

nicls67

### 5.29.2 Macro Definition Documentation

#### 5.29.2.1 BACKLIGHT\_PIN

```
#define BACKLIGHT_PIN 3
```

Backlight pin is on P3

Definition at line 17 of file LCD.h.

#### 5.29.2.2 EN\_PIN

```
#define EN_PIN 2
```

EN bit is on P2

Definition at line 14 of file LCD.h.

#### 5.29.2.3 LCD\_CNF\_BACKLIGHT\_OFF

```
#define LCD_CNF_BACKLIGHT_OFF 0
```

Backlight is disabled

Definition at line 70 of file LCD.h.

#### 5.29.2.4 LCD\_CNF\_BACKLIGHT\_ON

```
#define LCD_CNF_BACKLIGHT_ON 1
```

Backlight is enabled

Definition at line 69 of file LCD.h.

#### 5.29.2.5 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.

#### 5.29.2.6 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.

#### 5.29.2.7 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.

#### 5.29.2.8 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.

### 5.29.2.9 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.

### 5.29.2.10 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.

### 5.29.2.11 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.

### 5.29.2.12 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.

### 5.29.2.13 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.

**5.29.2.14 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.

**5.29.2.15 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.

**5.29.2.16 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.

**5.29.2.17 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.

**5.29.2.18 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.

### 5.29.2.19 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.

### 5.29.2.20 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.

### 5.29.2.21 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.

### 5.29.2.22 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.

### 5.29.2.23 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.

**5.29.2.24 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.

**5.29.2.25 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.

**5.29.2.26 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.

**5.29.2.27 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.

**5.29.2.28 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.

### 5.29.2.29 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.

### 5.29.2.30 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.

### 5.29.2.31 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.

### 5.29.2.32 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.

### 5.29.2.33 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.

**5.29.2.34 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.

**5.29.2.35 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.

**5.29.2.36 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.

**5.29.2.37 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.

**5.29.2.38 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.

### 5.29.2.39 LCD\_SIZE\_NB\_LINES

```
#define LCD_SIZE_NB_LINES 4
```

LCD screen has 4 lines

Definition at line 86 of file LCD.h.

### 5.29.2.40 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.

### 5.29.2.41 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.

### 5.29.2.42 RS\_PIN

```
#define RS_PIN 0
```

RS pin is on P0

Definition at line 16 of file LCD.h.

### 5.29.2.43 RW\_PIN

```
#define RW_PIN 1
```

RW pin is on P1

Definition at line 15 of file LCD.h.

## 5.29.3 Enumeration Type Documentation

### 5.29.3.1 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.

### 5.29.3.2 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.

### 5.29.3.3 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.

## 5.29.4 Variable Documentation

#### 5.29.4.1 p\_global\_BSW\_lcd

`LCD* p_global_BSW_lcd`

Pointer to `LCD` driver object

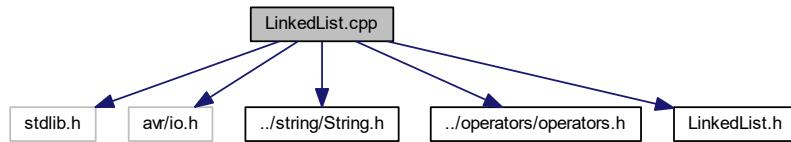
Definition at line 16 of file `LCD.cpp`.

## 5.30 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 "LinkedList.h"
```

Include dependency graph for `LinkedList.cpp`:



### 5.30.1 Detailed Description

Linked List library source file.

Date

27 avr. 2019

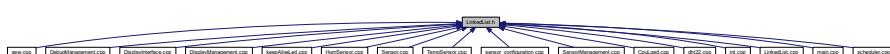
Author

nicls67

## 5.31 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`)

### 5.31.1 Detailed Description

Linked List library header file.

#### Date

27 avr. 2019

#### Author

nicls67

### 5.31.2 Typedef Documentation

#### 5.31.2.1 CompareFctPtr\_t

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

Type defining a pointer to the comparison function

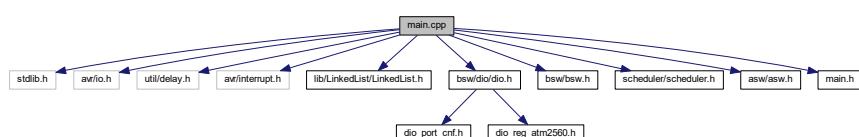
Definition at line 14 of file `LinkedList.h`.

## 5.32 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`

### 5.32.1 Detailed Description

Background task file.

#### Date

12 mars 2018

#### Author

nicls67

### 5.32.2 Macro Definition Documentation

#### 5.32.2.1 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.

### 5.32.3 Function Documentation

### 5.32.3.1 main()

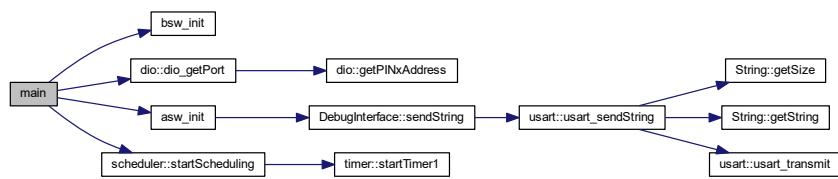
```
int main (
    void )
```

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:



### 5.32.4 Variable Documentation

#### 5.32.4.1 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.

#### 5.32.4.2 isDebugEnabled

```
bool isDebugEnabled
```

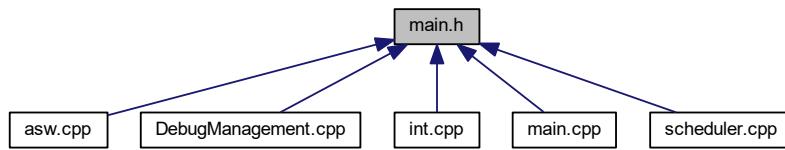
Flag indicating if the debug mode is activated or not

Definition at line 28 of file main.cpp.

## 5.33 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`

#### 5.33.1 Detailed Description

Background task header file.

##### Date

17 mars 2018

##### Author

nicls67

#### 5.33.2 Variable Documentation

##### 5.33.2.1 ASW\_init\_cnf

```
const T_ASW_init_cnf ASW_init_cnf
```

Definition of needed ASW services

Definition at line 30 of file main.cpp.

### 5.33.2.2 isDebugModeActivated

```
bool isDebugModeActivated
```

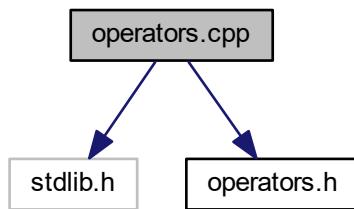
Flag indicating if the debug mode is activated or not

Definition at line 28 of file main.cpp.

## 5.34 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.*

### 5.34.1 Detailed Description

### c++ operators definitions

#### Date

14 mars 2018

#### Author

nicls67

## 5.34.2 Function Documentation

### 5.34.2.1 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.

### 5.34.2.2 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

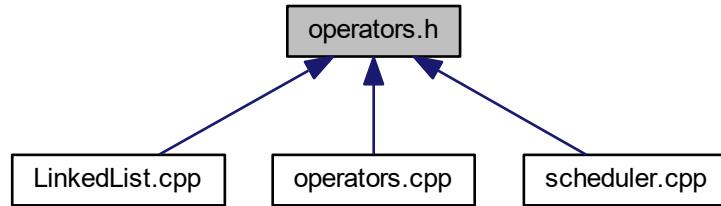
Pointer to the start of allocated memory zone

Definition at line 13 of file operators.cpp.

## 5.35 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.*

### 5.35.1 Detailed Description

c++ operators definitions header file

#### Date

14 mars 2018

#### Author

nicls67

### 5.35.2 Function Documentation

#### 5.35.2.1 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.

**5.35.2.2 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

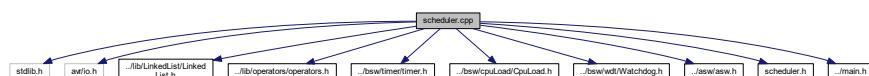
Definition at line 13 of file operators.cpp.

**5.36 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`

### 5.36.1 Detailed Description

Defines scheduler class.

#### Date

16 mars 2018

#### Author

nicls67

### 5.36.2 Variable Documentation

#### 5.36.2.1 `p_global_scheduler`

`scheduler* p_global_scheduler`

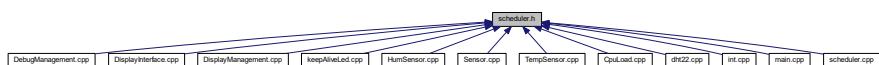
Pointer to scheduler object

Definition at line 27 of file scheduler.cpp.

## 5.37 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`

### 5.37.1 Detailed Description

Scheduler class header file.

#### Date

16 mars 2018

#### Author

nicls67

### 5.37.2 Macro Definition Documentation

#### 5.37.2.1 PRESCALER\_PERIODIC\_TIMER

```
#define PRESCALER_PERIODIC_TIMER 256
```

Value of prescaler to use for periodic timer

Definition at line 16 of file scheduler.h.

#### 5.37.2.2 SW\_PERIOD\_MS

```
#define SW_PERIOD_MS 500
```

Software period, used to define periodic timer interrupt

Definition at line 15 of file scheduler.h.

### 5.37.2.3 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.

### 5.37.3 Typedef Documentation

#### 5.37.3.1 TaskPtr\_t

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

Type defining a pointer to function.

Definition at line 22 of file scheduler.h.

### 5.37.4 Variable Documentation

#### 5.37.4.1 p\_global\_scheduler

```
scheduler* p_global_scheduler
```

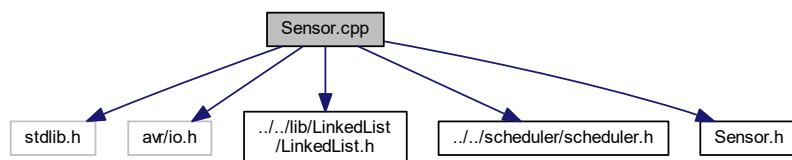
Pointer to scheduler object

Definition at line 27 of file scheduler.cpp.

## 5.38 Sensor.cpp File Reference

[Sensor](#) class source code file.

```
#include <stdlib.h>
#include <avr/io.h>
#include "../../lib/LinkedList/LinkedList.h"
#include "../../scheduler/scheduler.h"
#include "Sensor.h"
Include dependency graph for Sensor.cpp:
```



## Macros

- `#define TASK_PERIOD_DEFAULT 1000`
- `#define VALIDITY_TIMEOUT_MS_DEFAULT 30000`

### 5.38.1 Detailed Description

[Sensor](#) class source code file.

Date

20 juin 2019

Author

nicls67

### 5.38.2 Macro Definition Documentation

#### 5.38.2.1 TASK\_PERIOD\_DEFAULT

```
#define TASK_PERIOD_DEFAULT 1000
```

Default sensor task period : 1s

Definition at line 18 of file Sensor.cpp.

#### 5.38.2.2 VALIDITY\_TIMEOUT\_MS\_DEFAULT

```
#define VALIDITY_TIMEOUT_MS_DEFAULT 30000
```

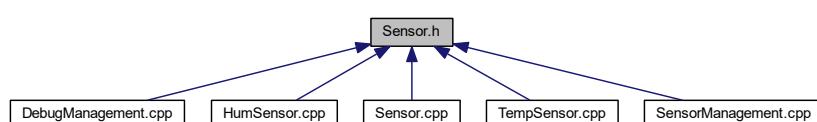
[Sensor](#) data are declared invalid after 30s

Definition at line 19 of file Sensor.cpp.

## 5.39 Sensor.h File Reference

[Sensor](#) class header file.

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



## Classes

- class [Sensor](#)

*Generic class for sensor device.*

### 5.39.1 Detailed Description

[Sensor](#) class header file.

#### Date

20 juin 2019

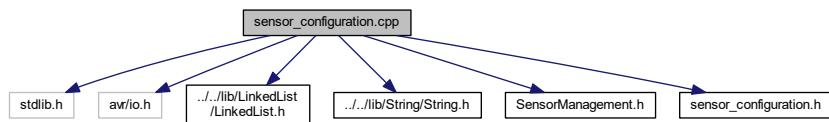
#### Author

nicls67

## 5.40 sensor\_configuration.cpp File Reference

[Sensor](#) configuration file.

```
#include <stdlib.h>
#include <avr/io.h>
#include "../../lib/LinkedList/LinkedList.h"
#include "../../lib/String/String.h"
#include "SensorManagement.h"
#include "sensor_configuration.h"
Include dependency graph for sensor_configuration.cpp:
```



## Macros

- `#define SENSOR_MGT_CNF_DEFAULT_PERIOD 1000`
- `#define SENSOR_MGT_CNF_DEFAULT_TMO 15000`

## Variables

- `T_SensorManagement_Sensor_Config SensorManagement_Sensor_Config_list [2]`  
*Sensor configuration table.*

### 5.40.1 Detailed Description

[Sensor](#) configuration file.

Date

22 juin 2019

Author

nicls67

### 5.40.2 Macro Definition Documentation

#### 5.40.2.1 SENSOR\_MGT\_CNF\_DEFAULT\_PERIOD

```
#define SENSOR_MGT_CNF_DEFAULT_PERIOD 1000
```

Default period for sensors task

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

#### 5.40.2.2 SENSOR\_MGT\_CNF\_DEFAULT\_TMO

```
#define SENSOR_MGT_CNF_DEFAULT_TMO 15000
```

Default timeout value for sensors

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

### 5.40.3 Variable Documentation

### 5.40.3.1 SensorManagement\_Sensor\_Config\_list

```
T_SensorManagement_Sensor_Config SensorManagement_Sensor_Config_list[2]
```

**Initial value:**

```
=
{
    {
        TEMPERATURE,
        SENSOR_MGT_CNF_DEFAULT_PERIOD,
        SENSOR_MGT_CNF_DEFAULT_TMO,
        (uint8_t*) "Temperature",
        (uint8_t*) "degC"
    },
    {
        HUMIDITY,
        SENSOR_MGT_CNF_DEFAULT_PERIOD,
        SENSOR_MGT_CNF_DEFAULT_TMO,
        (uint8_t*) "Humidite",
        (uint8_t*) "%"
    }
}
```

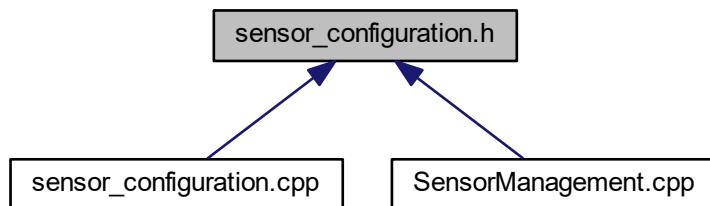
Sensor configuration table.

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

## 5.41 sensor\_configuration.h File Reference

Sensors configuration header file.

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



### Classes

- struct `T_SensorManagement_Sensor_Config`  
*Sensor informations structure.*

### Variables

- `T_SensorManagement_Sensor_Config SensorManagement_Sensor_Config_list [2]`  
*Sensor configuration table.*

### 5.41.1 Detailed Description

Sensors configuration header file.

#### Date

22 juin 2019

#### Author

nicls67

### 5.41.2 Variable Documentation

#### 5.41.2.1 SensorManagement\_Sensor\_Config\_list

`T_SensorManagement_Sensor_Config` `SensorManagement_Sensor_Config_list[2]`

**Sensor** configuration table.

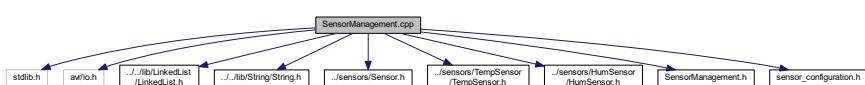
Definition at line 25 of file `sensor_configuration.cpp`.

## 5.42 SensorManagement.cpp File Reference

**SensorManagement** class source code file.

```
#include <stdlib.h>
#include <avr/io.h>
#include "../lib/LinkedList/LinkedList.h"
#include "../lib/String/String.h"
#include "../sensors/Sensor.h"
#include "../sensors/TempSensor/TempSensor.h"
#include "../sensors/HumSensor/HumSensor.h"
#include "SensorManagement.h"
#include "sensor_configuration.h"
```

Include dependency graph for `SensorManagement.cpp`:



## Variables

- `SensorManagement * p_global_ASW_SensorManagement`

### 5.42.1 Detailed Description

[SensorManagement](#) class source code file.

Date

22 juin 2019

Author

nicls67

### 5.42.2 Variable Documentation

#### 5.42.2.1 p\_global\_ASW\_SensorManagement

[SensorManagement](#)\* p\_global\_ASW\_SensorManagement

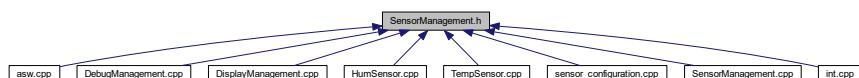
Pointer to the [SensorManagement](#) object

Definition at line 23 of file [SensorManagement.cpp](#).

## 5.43 SensorManagement.h File Reference

[SensorManagement](#) class header file.

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



## Classes

- class [SensorManagement](#)

*Sensor management class.*

## Enumerations

- enum [T\\_SensorManagement\\_Sensor\\_Type](#) { `TEMPERATURE`, `HUMIDITY` }

*Sensor type enumeration.*

## Variables

- `SensorManagement * p_global_ASW_SensorManagement`

### 5.43.1 Detailed Description

`SensorManagement` class header file.

#### Date

22 juin 2019

#### Author

nicls67

### 5.43.2 Enumeration Type Documentation

#### 5.43.2.1 `T_SensorManagement_Sensor_Type`

enum `T_SensorManagement_Sensor_Type`

`Sensor` type enumeration.

This enumeration defines all types of sensors available.

#### Enumerator

TEMPERATURE	
HUMIDITY	

Definition at line 17 of file `SensorManagement.h`.

### 5.43.3 Variable Documentation

#### 5.43.3.1 `p_global_ASW_SensorManagement`

`SensorManagement* p_global_ASW_SensorManagement`

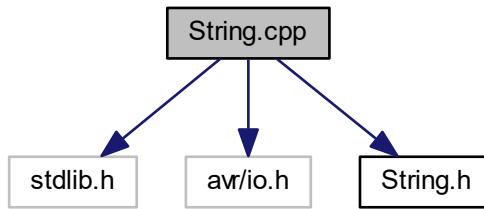
Pointer to the `SensorManagement` object

Definition at line 23 of file `SensorManagement.cpp`.

## 5.44 String.cpp File Reference

[String](#) class source file.

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



### 5.44.1 Detailed Description

[String](#) class source file.

Date

2 mai 2019

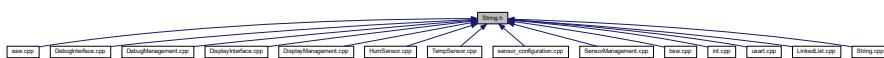
Author

nicls67

## 5.45 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.*

### 5.45.1 Detailed Description

[String](#) class header file.

Date

2 mai 2019

Author

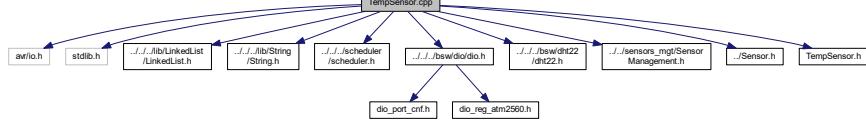
nicls67

## 5.46 TempSensor.cpp File Reference

Defines function of class [TempSensor](#).

```
#include <avr/io.h>
#include <stdlib.h>
#include "../../lib/LinkedList/LinkedList.h"
#include "../../lib/String/String.h"
#include "../../scheduler/scheduler.h"
#include "../../bsw/dio/dio.h"
#include "../../bsw/dht22/dht22.h"
#include "../../sensors_mgt/SensorManagement.h"
#include "Sensor.h"
#include "TempSensor.h"
```

Include dependency graph for TempSensor.cpp:



### Macros

- [#define DHT22\\_PORT ENCODE\\_PORT\(PORT\\_B, 6\)](#)

### 5.46.1 Detailed Description

Defines function of class [TempSensor](#).

Date

23 mars 2018

Author

nicls67

### 5.46.2 Macro Definition Documentation

#### 5.46.2.1 DHT22\_PORT

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

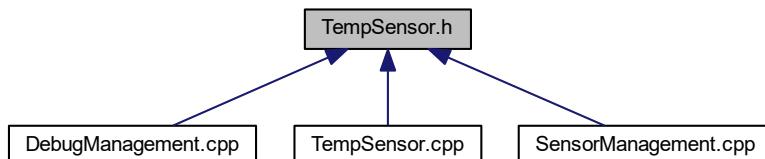
DHT22 is connected to port PB6

Definition at line 24 of file TempSensor.cpp.

## 5.47 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.*

### 5.47.1 Detailed Description

Class [TempSensor](#) header file.

#### Date

23 mars 2018

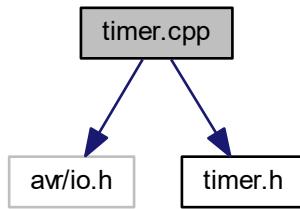
#### Author

nicls67

## 5.48 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`

#### 5.48.1 Detailed Description

Defines function for class timer.

##### Date

15 mars 2018

##### Author

nicls67

#### 5.48.2 Variable Documentation

##### 5.48.2.1 `p_global_BSW_timer`

`timer* p_global_BSW_timer`

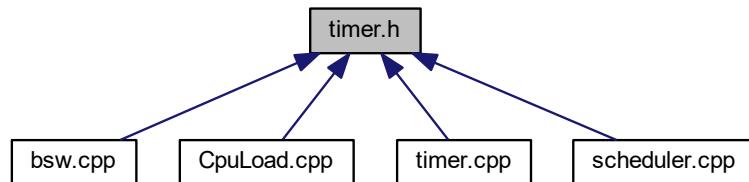
Pointer to timer driver object

Definition at line 13 of file timer.cpp.

## 5.49 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](#)

#### 5.49.1 Detailed Description

Timer class header file.

##### Date

15 mars 2018

##### Author

nicls67

#### 5.49.2 Variable Documentation

##### 5.49.2.1 [p\\_global\\_BSW\\_timer](#)

[timer](#)\* p\_global\_BSW\_timer

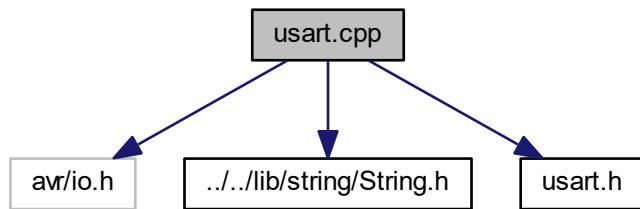
Pointer to timer driver object

Definition at line 13 of file timer.cpp.

## 5.50 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

- `uart * p_global_BSW_usart`

#### 5.50.1 Detailed Description

BSW library for USART.

##### Date

13 mars 2018

##### Author

nicls67

#### 5.50.2 Variable Documentation

##### 5.50.2.1 `p_global_BSW_usart`

`uart * p_global_BSW_usart`

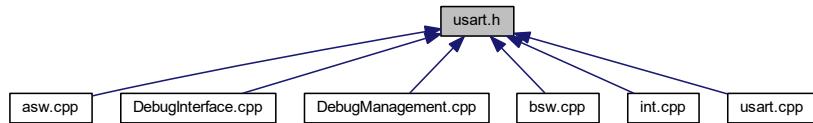
Pointer to usart driver object

Definition at line 16 of file usart.cpp.

## 5.51 usart.h File Reference

Header file for USART library.

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



### Classes

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

### Variables

- [uart \\* p\\_global\\_BSW\\_usart](#)

#### 5.51.1 Detailed Description

Header file for USART library.

##### Date

13 mars 2018

##### Author

nicls67

#### 5.51.2 Variable Documentation

##### 5.51.2.1 [p\\_global\\_BSW\\_usart](#)

[uart](#)\* p\_global\_BSW\_usart

Pointer to usart driver object

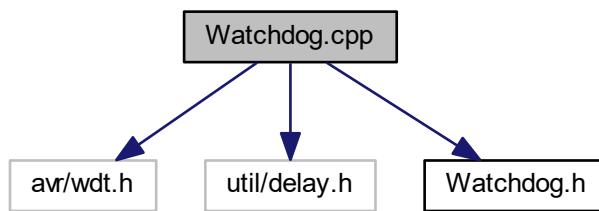
Definition at line 16 of file [usart.cpp](#).

## 5.52 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`

#### 5.52.1 Detailed Description

Class [Watchdog](#) source code file.

##### Date

6 juin 2019

##### Author

nicls67

#### 5.52.2 Macro Definition Documentation

### 5.52.2.1 WDG\_TIMEOUT\_DEFAULT\_MS

```
#define WDG_TIMEOUT_DEFAULT_MS WDG_TMO_500MS
```

Default timeout value is set to 500 ms

Definition at line 15 of file Watchdog.cpp.

### 5.52.3 Variable Documentation

#### 5.52.3.1 p\_global\_BSW\_wdg

```
Watchdog* p_global_BSW_wdg
```

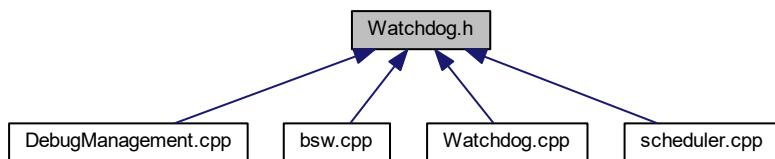
Pointer to [Watchdog](#) driver object

Definition at line 17 of file Watchdog.cpp.

## 5.53 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`

### 5.53.1 Detailed Description

Class `Watchdog` header file.

#### Date

6 juin 2019

#### Author

nicls67

### 5.53.2 Macro Definition Documentation

#### 5.53.2.1 WDG\_TMO\_120MS

```
#define WDG_TMO_120MS WDTO_120MS
```

Timeout value is 120 ms

Definition at line 19 of file Watchdog.h.

### 5.53.2.2 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.

### 5.53.2.3 WDG\_TMO\_1S

```
#define WDG_TMO_1S WDTO_1S
```

Timeout value is 1 s

Definition at line 22 of file Watchdog.h.

### 5.53.2.4 WDG\_TMO\_250MS

```
#define WDG_TMO_250MS WDTO_250MS
```

Timeout value is 250 ms

Definition at line 20 of file Watchdog.h.

### 5.53.2.5 WDG\_TMO\_2S

```
#define WDG_TMO_2S WDTO_2S
```

Timeout value is 2 s

Definition at line 23 of file Watchdog.h.

### 5.53.2.6 WDG\_TMO\_30MS

```
#define WDG_TMO_30MS WDTO_30MS
```

Timeout value is 30 ms

Definition at line 17 of file Watchdog.h.

### 5.53.2.7 WDG\_TMO\_4S

```
#define WDG_TMO_4S WDTO_4S
```

Timeout value is 4 s

Definition at line 24 of file Watchdog.h.

### 5.53.2.8 WDG\_TMO\_500MS

```
#define WDG_TMO_500MS WDTO_500MS
```

Timeout value is 500 ms

Definition at line 21 of file Watchdog.h.

### 5.53.2.9 WDG\_TMO\_60MS

```
#define WDG_TMO_60MS WDTO_60MS
```

Timeout value is 60 ms

Definition at line 18 of file Watchdog.h.

### 5.53.2.10 WDG\_TMO\_8S

```
#define WDG_TMO_8S WDTO_8S
```

Timeout value is 8 s

Definition at line 25 of file Watchdog.h.

## 5.53.3 Variable Documentation

### 5.53.3.1 p\_global\_BSW\_wdg

`Watchdog*` p\_global\_BSW\_wdg

Pointer to `Watchdog` driver object

Definition at line 17 of file Watchdog.cpp.

# Index

~LinkedList  
    LinkedList, 97  
~String  
    String, 127  
  
ASW\_init\_cnf  
    main.cpp, 231  
    main.h, 232  
addPeriodicTask  
    scheduler, 104  
alignment  
    T\_display\_data, 137  
appendBool  
    String, 128  
appendChar  
    String, 128  
appendInteger  
    String, 129  
appendSpace  
    String, 130  
appendString  
    String, 131  
asw.cpp, 169  
    asw\_init, 170  
asw.h, 171  
    asw\_init, 171  
asw\_init  
    asw.cpp, 170  
    asw.h, 171  
AttachNewElement  
    LinkedList, 98  
avg\_load  
    CpuLoad, 12  
  
BACKLIGHT\_PIN  
    LCD.h, 218  
backlight\_en  
    T\_LCD\_conf\_struct, 140  
backlight\_enable  
    LCD, 93  
BaudRate  
    uart, 159  
bitrate  
    I2C, 76  
blinkLed\_task  
    keepAliveLed, 77  
bsw.cpp, 172  
    bsw\_init, 173  
bsw.h, 174  
    bsw\_init, 174  
  
bsw\_init  
    bsw.cpp, 173  
    bsw.h, 174  
  
Clear  
    String, 132  
ClearFullScreen  
    DisplayInterface, 52  
ClearLine  
    DisplayInterface, 53  
ClearScreen  
    DebugInterface, 16  
ClearStringInDataStruct  
    DisplayInterface, 54  
cnfCursorBlink  
    LCD, 93  
cnfCursorOnOff  
    LCD, 93  
cnfDisplayOnOff  
    LCD, 93  
cnfEntryModeDir  
    LCD, 93  
cnfEntryModeShift  
    LCD, 93  
cnfFontType  
    LCD, 94  
cnfI2C\_addr  
    LCD, 94  
cnfLineNumber  
    LCD, 94  
command  
    LCD, 81  
CompareFctPtr\_t  
    LinkedList.h, 229  
ComputeCPULoad  
    CpuLoad, 10  
ComputeStringSize  
    String, 132  
ConfigureBacklight  
    LCD, 82  
ConfigureCursorBlink  
    LCD, 82  
ConfigureCursorOnOff  
    LCD, 83  
ConfigureDisplayOnOff  
    LCD, 84  
ConfigureEntryModeDir  
    LCD, 84  
ConfigureEntryModeShift  
    LCD, 85

ConfigureFontType  
 LCD, 86  
 ConfigureI2CAddr  
 LCD, 86  
 ConfigureLineNumber  
 LCD, 87  
 configureTimer1  
 timer, 151  
 CpuLoad, 9  
 avg\_load, 12  
 ComputeCPULoad, 10  
 CpuLoad, 10  
 current\_load, 12  
 getAverageCPULoad, 11  
 getCurrentCPULoad, 11  
 getMaxCPULoad, 11  
 last\_sum\_value, 12  
 max\_load, 13  
 sample\_cnt, 13  
 sample\_idx, 13  
 sample\_mem, 13  
 CpuLoad.cpp, 175  
 p\_global\_BSW\_cpuload, 176  
 CpuLoad.h, 176  
 NB\_OF\_SAMPLES, 177  
 p\_global\_BSW\_cpuload, 177  
 curElement\_ptr  
 LinkedList, 102  
 current\_load  
 CpuLoad, 12  
 cursor\_en  
 T\_LCD\_conf\_struct, 140  
 cursorBlink\_en  
 T\_LCD\_conf\_struct, 140  
 DATA\_ACK  
 I2C.h, 209  
 DDRA\_PTR  
 dio\_reg\_atm2560.h, 195  
 DDRB\_PTR  
 dio\_reg\_atm2560.h, 196  
 DDRC\_PTR  
 dio\_reg\_atm2560.h, 196  
 DDRD\_PTR  
 dio\_reg\_atm2560.h, 196  
 DEBUG\_ACTIVE\_PORT  
 main.cpp, 230  
 DECODE\_PIN  
 dio.h, 191  
 DECODE\_PORT  
 dio.h, 191  
 DHT22\_PORT  
 HumSensor.cpp, 206  
 TempSensor.cpp, 249  
 DISPLAY\_LINE\_SHIFT\_PERIOD\_MS  
 DisplayInterface.h, 200  
 DISPLAY\_LINE\_SHIFT\_TEMPO\_TIME  
 DisplayInterface.h, 200  
 DISPLAY\_MGT\_FIRST\_LINE\_SENSORS  
 DisplayManagement.h, 204  
 DISPLAY\_MGT\_I2C\_BITRATE  
 DisplayManagement.h, 204  
 DISPLAY\_MGT\_LCD\_I2C\_ADDR  
 DisplayManagement.h, 204  
 DISPLAY\_MGT\_PERIOD\_TASK\_SENSOR  
 DisplayManagement.h, 204  
 DISPLAY\_MGT\_PERIOD\_WELCOME\_MSG\_REMOTE  
 VAL  
 DisplayManagement.h, 205  
 data\_name\_str  
 T\_SensorManagement\_Sensor\_Config, 144  
 data\_ptr  
 LinkedList::T\_LL\_element, 143  
 ddram\_addr  
 LCD, 94  
 debug\_ift\_ptr  
 DebugManagement, 32  
 debug\_mgt\_main\_menu\_state\_t  
 DebugManagement.h, 185  
 debug\_mgt\_state\_struct\_t, 14  
 main\_state, 14  
 wdg\_state, 14  
 debug\_mgt\_wdg\_state\_t  
 DebugManagement.h, 185  
 debug\_state  
 DebugManagement, 32  
 DebugInterface, 15  
 ClearScreen, 16  
 DebugInterface, 16  
 nextLine, 16  
 read, 17  
 sendBool, 17  
 sendChar, 18  
 sendInteger, 19  
 sendString, 20, 21  
 usart\_drv\_ptr, 21  
 DebugInterface.cpp, 178  
 p\_global\_ASW\_DebugInterface, 178  
 DebugInterface.h, 179  
 p\_global\_ASW\_DebugInterface, 180  
 USART\_BAUDRATE, 179  
 DebugManagement, 22  
 debug\_ift\_ptr, 32  
 debug\_state, 32  
 DebugManagement, 23  
 DebugModeManagement, 24  
 DisplayData, 25  
 DisplayPeriodicData\_task, 26  
 exitDebugMenu, 27  
 getIftPtr, 28  
 getInfoStringPtr, 28  
 getMenuStringPtr, 28  
 info\_string\_ptr, 32  
 isInfoStringDisplayed, 33  
 MainMenuManagement, 29  
 menu\_string\_ptr, 33  
 sensorMgt\_ptr, 33

setInfoStringPtr, 30  
systemReset, 30  
WatchdogMenuManagement, 31  
DebugManagement.cpp, 180  
  p\_global\_ASW\_DebugManagement, 181  
  str\_debug\_info\_message\_wdg\_disabled, 181  
  str\_debug\_info\_message\_wdg\_enabled, 182  
  str\_debug\_info\_message\_wdg\_tmo\_updated, 182  
  str\_debug\_info\_message\_wdg\_tmo\_value, 182  
  str\_debug\_info\_message\_wrong\_menu\_selection, 182  
  str\_debug\_main\_menu, 182  
  str\_debug\_wdg\_menu, 183  
  str\_debug\_wdg\_timeout\_update\_selection, 183  
DebugManagement.h, 184  
  debug\_mgt\_main\_menu\_state\_t, 185  
  debug\_mgt\_wdg\_state\_t, 185  
  p\_global\_ASW\_DebugManagement, 186  
  PERIOD\_MS\_TASK\_DISPLAY\_CPU\_LOAD, 185  
  PERIOD\_MS\_TASK\_DISPLAY\_DEBUG\_DATA, 185  
DebugModeManagement  
  DebugManagement, 24  
dht22, 34  
  dht22, 35  
  dht22\_port, 40  
  dio\_ptr, 40  
  getHumidity, 35  
  getTemperature, 37  
  initializeCommunication, 38  
  mem\_humidity, 40  
  mem\_temperature, 40  
  mem\_validity, 40  
  pit\_last\_read, 40  
  read, 39  
dht22.cpp, 186  
  MAX\_WAIT\_TIME\_US, 187  
  p\_global\_BSW\_dht22, 187  
dht22.h, 188  
  p\_global\_BSW\_dht22, 188  
dht22\_port  
  dht22, 40  
dio, 41  
  dio, 42  
  dio\_changePortPinCnf, 42  
  dio\_getPort, 43  
  dio\_getPort\_fast, 44  
  dio\_invertPort, 44  
  dio\_memorizePINaddress, 45  
  dio\_setPort, 46  
  getDDRxAddress, 47  
  getPINxAddress, 48  
  getPORTxAddress, 48  
  PINx\_addr\_mem, 49  
  PINx\_idx\_mem, 50  
  ports\_init, 49  
dio.cpp, 189  
  p\_global\_BSW\_dio, 189  
dio.h, 190  
  DECODE\_PIN, 191  
  DECODE\_PORT, 191  
  ENCODE\_PORT, 191  
  p\_global\_BSW\_dio, 192  
  PORT\_CNF\_IN, 191  
  PORT\_CNF\_OUT, 192  
dio\_changePortPinCnf  
  dio, 42  
dio\_getPort  
  dio, 43  
dio\_getPort\_fast  
  dio, 44  
dio\_invertPort  
  dio, 44  
dio\_memorizePINaddress  
  dio, 45  
dio\_port\_cnf.h, 192  
  PORT\_A, 193  
  PORT\_B, 193  
  PORT\_C, 193  
  PORT\_D, 194  
  PORTB\_CNF\_DDRB, 194  
  PORTB\_CNF\_PORTB, 194  
dio\_ptr  
  dht22, 40  
dio\_reg\_atm2560.h, 195  
  DDRA\_PTR, 195  
  DDR\_B\_PTR, 196  
  DDRC\_PTR, 196  
  DDR\_D\_PTR, 196  
  PINA\_PTR, 196  
  PINB\_PTR, 196  
  PINC\_PTR, 197  
  PIN\_D\_PTR, 197  
  PORTA\_PTR, 197  
  PORTB\_PTR, 197  
  PORTC\_PTR, 197  
  PORTD\_PTR, 198  
dio\_setPort  
  dio, 46  
disable  
  Watchdog, 161  
display\_data  
  DisplayInterface, 62  
display\_en  
  T\_LCD\_conf\_struct, 141  
display\_str  
  T\_display\_data, 137  
DisplayData  
  DebugManagement, 25  
DisplayFullLine  
  DisplayInterface, 55, 56  
DisplayInterface, 50  
  ClearFullScreen, 52  
  ClearLine, 53  
  ClearStringInDataStruct, 54  
  display\_data, 62

DisplayFullLine, 55, 56  
 DisplayInterface, 52  
 dummy, 62  
 FindFirstCharAddr, 56  
 getDisplayDataPtr, 57  
 IsLineEmpty, 58  
 isShiftInProgress, 62  
 p\_lcd, 62  
 RefreshLine, 58  
 setLineAlignment, 59  
 setLineAlignmentAndRefresh, 60  
 shiftLine\_task, 60  
 updateLineAndRefresh, 61  
 DisplayInterface.cpp, 198  
 p\_global\_ASW\_DisplayInterface, 199  
 DisplayInterface.h, 199  
 DISPLAY\_LINE\_SHIFT\_PERIOD\_MS, 200  
 DISPLAY\_LINE\_SHIFT\_TEMPO\_TIME, 200  
 p\_global\_ASW\_DisplayInterface, 201  
 T\_DisplayInterface\_LineAlignment, 200  
 T\_DisplayInterface\_LineDisplayMode, 201  
 DisplayManagement, 63  
 DisplayManagement, 64  
 DisplaySensorData\_Task, 65  
 GetIftPointer, 65  
 GetSensorMgtPtr, 66  
 p\_SensorMgt, 67  
 p\_display\_ift, 67  
 RemoveWelcomeMessage\_Task, 66  
 DisplayManagement.cpp, 202  
 noSensorsDisplayString, 202  
 p\_global\_ASW\_DisplayManagement, 202  
 welcomeMessageString, 203  
 DisplayManagement.h, 203  
 DISPLAY\_MGT\_FIRST\_LINE\_SENSORS, 204  
 DISPLAY\_MGT\_I2C\_BITRATE, 204  
 DISPLAY\_MGT\_LCD\_I2C\_ADDR, 204  
 DISPLAY\_MGT\_PERIOD\_TASK\_SENSOR, 204  
 DISPLAY\_MGT\_PERIOD\_WELCOME\_MSG\_R←  
     EMOVAL, 205  
 LCD\_init\_cnf, 205  
 p\_global\_ASW\_DisplayManagement, 205  
 DisplayPeriodicData\_task  
     DebugManagement, 26  
 DisplaySensorData\_Task  
     DisplayManagement, 65  
 dummy  
     DisplayInterface, 62  
  
 EN\_PIN  
     LCD.h, 218  
 ENCODE\_PORT  
     dio.h, 191  
 enable  
     Watchdog, 162  
 entryModeDir  
     T\_LCD\_conf\_struct, 141  
 entryModeShift  
     T\_LCD\_conf\_struct, 141  
  
 exitDebugMenu  
     DebugManagement, 27  
  
 FindElement  
     LinkedList, 98  
 FindFirstCharAddr  
     DisplayInterface, 56  
 firstElement  
     LinkedList, 102  
 fontType\_cnf  
     T\_LCD\_conf\_struct, 141  
  
 getAverageCPUload  
     CpuLoad, 11  
 getCurrentElement  
     LinkedList, 99  
 getCurrentCPUload  
     CpuLoad, 11  
 GetDDRAMAddress  
     LCD, 88  
 getDDRxAddress  
     dio, 47  
 getDisplayDataPtr  
     DisplayInterface, 57  
 getFullStringFormattedValue  
     SensorManagement, 121  
 getHumidity  
     dht22, 35  
 GetIftPointer  
     DisplayManagement, 65  
 getIftPtr  
     DebugManagement, 28  
 getInfoStringPtr  
     DebugManagement, 28  
 GetLineNumberCnf  
     LCD, 88  
 getMaxCPUload  
     CpuLoad, 11  
 getMenuStringPtr  
     DebugManagement, 28  
 getPINxAddress  
     dio, 48  
 getPORTxAddress  
     dio, 48  
 getPitNumber  
     scheduler, 105  
 getRawDataPtr  
     Sensor, 114  
 getSensorCount  
     SensorManagement, 122  
 GetSensorMgtPtr  
     DisplayManagement, 66  
 getSensorObjectPtr  
     SensorManagement, 122  
 getSize  
     String, 133  
 getString  
     String, 133  
 getTMOValue

Watchdog, 162  
getTaskCount  
    scheduler, 106  
getTaskPeriod  
    Sensor, 114  
getTemperature  
    dht22, 37  
getTimer1Value  
    timer, 152  
getValidity  
    Sensor, 115  
getValue  
    Sensor, 115  
getValueDecimal  
    Sensor, 115  
getValueInteger  
    Sensor, 116  
  
HumSensor, 68  
    HumSensor, 69  
    readHumSensor\_task, 70  
    updateTaskPeriod, 71  
HumSensor.cpp, 206  
    DHT22\_PORT, 206  
HumSensor.h, 207  
  
I2C.cpp, 207  
    p\_global\_BSW\_i2c, 208  
I2C.h, 208  
    DATA\_ACK, 209  
    p\_global\_BSW\_i2c, 210  
    SLA\_ACK, 209  
    START, 209  
I2C, 72  
    bitrate, 76  
    I2C, 73  
    initializeBus, 73  
    setBitRate, 74  
    write, 74  
    writeByte, 75  
i2c\_addr  
    T\_LCD\_conf\_struct, 141  
i2c\_bitrate  
    T\_LCD\_conf\_struct, 142  
i2c\_drv\_ptr  
    LCD, 94  
ISR  
    int.cpp, 211  
info\_string\_ptr  
    DebugManagement, 32  
initializeBus  
    I2C, 73  
initializeCommunication  
    dht22, 38  
InitializeScreen  
    LCD, 88  
int.cpp, 210  
    ISR, 211  
isActive  
    Watchdog, 166  
isDebugEnabled  
    T\_ASW\_init\_cnf, 135  
isDebugModeActivated  
    main.cpp, 231  
    main.h, 232  
isDisplayActivated  
    T\_ASW\_init\_cnf, 135  
isEmpty  
    T\_display\_data, 137  
isEnabled  
    Watchdog, 163  
isInfoStringDisplayed  
    DebugManagement, 33  
isLEDActivated  
    T\_ASW\_init\_cnf, 135  
IsLLEmpty  
    LinkedList, 99  
IsLineEmpty  
    DisplayInterface, 58  
isSensorMgtActivated  
    T\_ASW\_init\_cnf, 136  
isShiftInProgress  
    DisplayInterface, 62  
  
keepAliveLed, 76  
    blinkLed\_task, 77  
    keepAliveLed, 77  
keepAliveLed.cpp, 212  
    p\_global\_ASW\_keepAliveLed, 213  
keepAliveLed.h, 213  
    LED\_PORT, 214  
    p\_global\_ASW\_keepAliveLed, 215  
    PERIOD\_MS\_TASK\_LED, 214  
  
LCD.cpp, 215  
    p\_global\_BSW\_lcd, 216  
LCD.h, 216  
    BACKLIGHT\_PIN, 218  
    EN\_PIN, 218  
    LCD\_CNF\_BACKLIGHT\_OFF, 218  
    LCD\_CNF\_BACKLIGHT\_ON, 218  
    LCD\_CNF\_CURSOR\_BLINK\_OFF, 219  
    LCD\_CNF\_CURSOR\_BLINK\_ON, 219  
    LCD\_CNF\_CURSOR\_OFF, 219  
    LCD\_CNF\_CURSOR\_ON, 219  
    LCD\_CNF\_DISPLAY\_OFF, 219  
    LCD\_CNF\_DISPLAY\_ON, 220  
    LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_LEFT,  
        220  
    LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_RIGHT,  
        220  
    LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT\_↔  
        OFF, 220  
    LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT\_↔  
        ON, 220  
    LCD\_CNF\_FONT\_5\_11, 221  
    LCD\_CNF\_FONT\_5\_8, 221  
    LCD\_CNF\_ONE\_LINE, 221

LCD\_CNF\_SHIFT\_ID, 221  
LCD\_CNF\_SHIFT\_SH, 221  
LCD\_CNF\_TWO\_LINE, 222  
LCD\_DISPLAY\_CTRL\_FIELD\_B, 222  
LCD\_DISPLAY\_CTRL\_FIELD\_C, 222  
LCD\_DISPLAY\_CTRL\_FIELD\_D, 222  
LCD\_FCT\_SET\_FIELD\_DL, 222  
LCD\_FCT\_SET\_FIELD\_F, 223  
LCD\_FCT\_SET\_FIELD\_N, 223  
LCD\_INST\_CLR\_DISPLAY\_BIT, 223  
LCD\_INST\_DISPLAY\_CTRL, 223  
LCD\_INST\_ENTRY\_MODE\_SET, 223  
LCD\_INST\_FUNCTION\_SET, 224  
LCD\_INST\_SET\_DDRAM\_ADDR, 224  
LCD\_RAM\_1\_LINE\_MAX, 224  
LCD\_RAM\_1\_LINE\_MIN, 224  
LCD\_RAM\_2\_LINES\_MAX\_1, 224  
LCD\_RAM\_2\_LINES\_MAX\_2, 225  
LCD\_RAM\_2\_LINES\_MIN\_1, 225  
LCD\_RAM\_2\_LINES\_MIN\_2, 225  
LCD\_SIZE\_NB\_CHAR\_PER\_LINE, 225  
LCD\_SIZE\_NB\_LINES, 225  
LCD\_WAIT\_CLR\_RETURN, 226  
LCD\_WAIT\_OTHER\_MODES, 226  
p\_global\_BSW\_lcd, 227  
RS\_PIN, 226  
RW\_PIN, 226  
T\_LCD\_command, 226  
T\_LCD\_config\_mode, 227  
T\_LCD\_ram\_area, 227  
LCD\_CNF\_BACKLIGHT\_OFF  
    LCD.h, 218  
LCD\_CNF\_BACKLIGHT\_ON  
    LCD.h, 218  
LCD\_CNF\_CURSOR\_BLINK\_OFF  
    LCD.h, 219  
LCD\_CNF\_CURSOR\_BLINK\_ON  
    LCD.h, 219  
LCD\_CNF\_CURSOR\_OFF  
    LCD.h, 219  
LCD\_CNF\_CURSOR\_ON  
    LCD.h, 219  
LCD\_CNF\_DISPLAY\_OFF  
    LCD.h, 219  
LCD\_CNF\_DISPLAY\_ON  
    LCD.h, 220  
LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_LEFT  
    LCD.h, 220  
LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_RIGHT  
    LCD.h, 220  
LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT\_OFF  
    LCD.h, 220  
LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT\_ON  
    LCD.h, 220  
LCD\_CNF\_FONT\_5\_11  
    LCD.h, 221  
LCD\_CNF\_FONT\_5\_8  
    LCD.h, 221  
LCD\_CNF\_ONE\_LINE  
    LCD.h, 221  
LCD\_CNF\_SHIFT\_ID  
    LCD.h, 221  
LCD\_CNF\_SHIFT\_SH  
    LCD.h, 221  
LCD\_CNF\_TWO\_LINE  
    LCD.h, 222  
LCD\_DISPLAY\_CTRL\_FIELD\_B  
    LCD.h, 222  
LCD\_DISPLAY\_CTRL\_FIELD\_C  
    LCD.h, 222  
LCD\_DISPLAY\_CTRL\_FIELD\_D  
    LCD.h, 222  
LCD\_FCT\_SET\_FIELD\_DL  
    LCD.h, 222  
LCD\_FCT\_SET\_FIELD\_F  
    LCD.h, 223  
LCD\_FCT\_SET\_FIELD\_N  
    LCD.h, 223  
LCD\_INST\_CLR\_DISPLAY\_BIT  
    LCD.h, 223  
LCD\_INST\_DISPLAY\_CTRL  
    LCD.h, 223  
LCD\_INST\_ENTRY\_MODE\_SET  
    LCD.h, 223  
LCD\_INST\_FUNCTION\_SET  
    LCD.h, 224  
LCD\_INST\_SET\_DDRAM\_ADDR  
    LCD.h, 224  
LCD\_RAM\_1\_LINE\_MAX  
    LCD.h, 224  
LCD\_RAM\_1\_LINE\_MIN  
    LCD.h, 224  
LCD\_RAM\_2\_LINES\_MAX\_1  
    LCD.h, 224  
LCD\_RAM\_2\_LINES\_MAX\_2  
    LCD.h, 225  
LCD\_RAM\_2\_LINES\_MIN\_1  
    LCD.h, 225  
LCD\_RAM\_2\_LINES\_MIN\_2  
    LCD.h, 225  
LCD\_SIZE\_NB\_CHAR\_PER\_LINE  
    LCD.h, 225  
LCD\_SIZE\_NB\_LINES  
    LCD.h, 225  
LCD\_WAIT\_CLR\_RETURN  
    LCD.h, 226  
LCD\_WAIT\_OTHER\_MODES  
    LCD.h, 226  
LCD\_init\_cnf  
    DisplayManagement.h, 205  
LCD, 78  
    backlight\_enable, 93  
    cnfCursorBlink, 93  
    cnfCursorOnOff, 93  
    cnfDisplayOnOff, 93  
    cnfEntryModeDir, 93

cnfEntryModeShift, 93  
cnfFontType, 94  
cnfI2C\_addr, 94  
cnfLineNumber, 94  
command, 81  
ConfigureBacklight, 82  
ConfigureCursorBlink, 82  
ConfigureCursorOnOff, 83  
ConfigureDisplayOnOff, 84  
ConfigureEntryModeDir, 84  
ConfigureEntryModeShift, 85  
ConfigureFontType, 86  
ConfigureI2CAddr, 86  
ConfigureLineNumber, 87  
ddram\_addr, 94  
GetDDRAMAddress, 88  
GetLineNumberCnf, 88  
i2c\_drv\_ptr, 94  
InitializeScreen, 88  
LCD, 80  
SetDDRAMAddress, 89  
write, 90  
write4bits, 91  
WriteInRam, 92  
LED\_PORT  
  keepAliveLed.h, 214  
LLElementCompare  
  scheduler, 107  
last\_sum\_value  
  CpuLoad, 12  
launchPeriodicTasks  
  scheduler, 106  
lineNumber\_cnf  
  T\_LCD\_conf\_struct, 142  
LinkedList, 95  
  ~LinkedList, 97  
  AttachNewElement, 98  
  curElement\_ptr, 102  
  FindElement, 98  
  firstElement, 102  
  getCurrentElement, 99  
  IsLLEmpty, 99  
  LinkedList, 97  
  MoveToNextElement, 100  
  RemoveElement, 100  
  ResetElementPtr, 101  
  T\_LL\_element, 96  
LinkedList.cpp, 228  
LinkedList.h, 228  
  CompareFctPtr\_t, 229  
LinkedList::T\_LL\_element, 142  
  data\_ptr, 143  
  nextElement, 143  
MAX\_WAIT\_TIME\_US  
  dht22.cpp, 187  
main  
  main.cpp, 230  
main.cpp, 229  
ASW\_init\_cnf, 231  
DEBUG\_ACTIVE\_PORT, 230  
isDebugModeActivated, 231  
  main, 230  
main.h, 232  
  ASW\_init\_cnf, 232  
  isDebugModeActivated, 232  
main\_state  
  debug\_mgt\_state\_struct\_t, 14  
MainMenuManagement  
  DebugManagement, 29  
max\_load  
  CpuLoad, 13  
mem\_humidity  
  dht22, 40  
mem\_temperature  
  dht22, 40  
mem\_validity  
  dht22, 40  
menu\_string\_ptr  
  DebugManagement, 33  
mode  
  T\_display\_data, 137  
MoveToNextElement  
  LinkedList, 100  
NB\_OF\_SAMPLES  
  CpuLoad.h, 177  
nb\_sensors  
  SensorManagement, 124  
nextElement  
  LinkedList::T\_LL\_element, 143  
nextLine  
  DebugInterface, 16  
noSensorsDisplayString  
  DisplayManagement.cpp, 202  
operator delete  
  operators.cpp, 234  
  operators.h, 235  
operator new  
  operators.cpp, 234  
  operators.h, 236  
operators.cpp, 233  
  operator delete, 234  
  operator new, 234  
operators.h, 234  
  operator delete, 235  
  operator new, 236  
p\_SensorMgt  
  DisplayManagement, 67  
p\_display\_ift  
  DisplayManagement, 67  
p\_global\_ASW\_DebugInterface  
  DebugInterface.cpp, 178  
  DebugInterface.h, 180  
p\_global\_ASW\_DebugManagement  
  DebugManagement.cpp, 181

DebugManagement.h, 186  
**p\_global\_ASW\_DisplayInterface**  
 DisplayInterface.cpp, 199  
 DisplayInterface.h, 201  
**p\_global\_ASW\_DisplayManagement**  
 DisplayManagement.cpp, 202  
 DisplayManagement.h, 205  
**p\_global\_ASW\_SensorManagement**  
 SensorManagement.cpp, 245  
 SensorManagement.h, 246  
**p\_global\_ASW\_keepAliveLed**  
 keepAliveLed.cpp, 213  
 keepAliveLed.h, 215  
**p\_global\_BSW\_cpupload**  
 CpuLoad.cpp, 176  
 CpuLoad.h, 177  
**p\_global\_BSW\_dht22**  
 dht22.cpp, 187  
 dht22.h, 188  
**p\_global\_BSW\_dio**  
 dio.cpp, 189  
 dio.h, 192  
**p\_global\_BSW\_i2c**  
 I2C.cpp, 208  
 I2C.h, 210  
**p\_global\_BSW\_lcd**  
 LCD.cpp, 216  
 LCD.h, 227  
**p\_global\_BSW\_timer**  
 timer.cpp, 250  
 timer.h, 251  
**p\_global\_BSW\_usart**  
 usart.cpp, 252  
 usart.h, 253  
**p\_global\_BSW\_wdg**  
 Watchdog.cpp, 255  
 Watchdog.h, 258  
**p\_global\_scheduler**  
 scheduler.cpp, 237  
 scheduler.h, 239  
**p\_lcd**  
 DisplayInterface, 62  
**PERIOD\_MS\_TASK\_DISPLAY\_CPU\_LOAD**  
 DebugManagement.h, 185  
**PERIOD\_MS\_TASK\_DISPLAY\_DEBUG\_DATA**  
 DebugManagement.h, 185  
**PERIOD\_MS\_TASK\_LED**  
 keepAliveLed.h, 214  
**PINA\_PTR**  
 dio\_reg\_atm2560.h, 196  
**PINB\_PTR**  
 dio\_reg\_atm2560.h, 196  
**PINC\_PTR**  
 dio\_reg\_atm2560.h, 197  
**PIND\_PTR**  
 dio\_reg\_atm2560.h, 197  
**PINx\_addr\_mem**  
 dio, 49  
**PINx\_idx\_mem**  
 dio, 50  
**PORT\_CNF\_IN**  
 dio.h, 191  
**PORT\_CNF\_OUT**  
 dio.h, 192  
**PORT\_A**  
 dio\_port\_cnf.h, 193  
**PORT\_B**  
 dio\_port\_cnf.h, 193  
**PORT\_C**  
 dio\_port\_cnf.h, 193  
**PORT\_D**  
 dio\_port\_cnf.h, 194  
**PORTA\_PTR**  
 dio\_reg\_atm2560.h, 197  
**PORTB\_CNF\_DDRB**  
 dio\_port\_cnf.h, 194  
**PORTB\_CNF\_PORTB**  
 dio\_port\_cnf.h, 194  
**PORTB\_PTR**  
 dio\_reg\_atm2560.h, 197  
**PORTC\_PTR**  
 dio\_reg\_atm2560.h, 197  
**PORTD\_PTR**  
 dio\_reg\_atm2560.h, 198  
**PRESCALER\_PERIODIC\_TIMER**  
 scheduler.h, 238  
**period**  
 scheduler::Task\_t, 145  
 T\_SensorManagement\_Sensor\_Config, 144  
**pit\_last\_read**  
 dht22, 40  
**pit\_number**  
 scheduler, 111  
**ports\_init**  
 dio, 49  
**prescaler**  
 timer, 153  
**RS\_PIN**  
 LCD.h, 226  
**RW\_PIN**  
 LCD.h, 226  
**raw\_data**  
 Sensor, 119  
**read**  
 DebugInterface, 17  
 dht22, 39  
**readHumSensor\_task**  
 HumSensor, 70  
**readSensor\_task**  
 Sensor, 116  
**readTempSensor\_task**  
 TempSensor, 148  
**RefreshLine**  
 DisplayInterface, 58  
**RemoveElement**  
 LinkedList, 100

removePeriodicTask  
scheduler, 108  
RemoveWelcomeMessage\_Task  
DisplayManagement, 66  
reset  
Watchdog, 163  
ResetElementPtr  
LinkedList, 101  
  
SENSOR\_MGT\_CNF\_DEFAULT\_PERIOD  
sensor\_configuration.cpp, 242  
SENSOR\_MGT\_CNF\_DEFAULT\_TMO  
sensor\_configuration.cpp, 242  
SLA\_ACK  
I2C.h, 209  
START  
I2C.h, 209  
SW\_PERIOD\_MS  
scheduler.h, 238  
sample\_cnt  
CpuLoad, 13  
sample\_idx  
CpuLoad, 13  
sample\_mem  
CpuLoad, 13  
scheduler, 102  
addPeriodicTask, 104  
getPitNumber, 105  
getTaskCount, 106  
LLElementCompare, 107  
launchPeriodicTasks, 106  
pit\_number, 111  
removePeriodicTask, 108  
scheduler, 104  
startScheduling, 109  
task\_count, 111  
Task\_t, 104  
TasksLL\_ptr, 111  
updateTaskPeriod, 110  
scheduler.cpp, 236  
p\_global\_scheduler, 237  
scheduler.h, 237  
p\_global\_scheduler, 239  
PRESCALER\_PERIODIC\_TIMER, 238  
SW\_PERIOD\_MS, 238  
TIMER\_CTC\_VALUE, 238  
TaskPtr\_t, 239  
scheduler::Task\_t, 145  
period, 145  
TaskPtr, 145  
sendBool  
DebugInterface, 17  
sendChar  
DebugInterface, 18  
sendInteger  
DebugInterface, 19  
sendString  
DebugInterface, 20, 21  
Sensor, 112  
getRawDataPtr, 114  
getTaskPeriod, 114  
getValidity, 115  
getValue, 115  
getValueDecimal, 115  
getValueInteger, 116  
raw\_data, 119  
readSensor\_task, 116  
Sensor, 113  
setLastValidity, 116  
setValidityTMO, 117  
task\_period, 119  
updateTaskPeriod, 117  
updateValidData, 118  
valid\_pit, 119  
valid\_value, 119  
validity, 119  
validity\_last\_read, 119  
validity\_tmo, 120  
Sensor.cpp, 239  
TASK\_PERIOD\_DEFAULT, 240  
VALIDITY\_TIMEOUT\_MS\_DEFAULT, 240  
Sensor.h, 240  
sensor\_configuration.cpp, 241  
SENSOR\_MGT\_CNF\_DEFAULT\_PERIOD, 242  
SENSOR\_MGT\_CNF\_DEFAULT\_TMO, 242  
SensorManagement\_Sensor\_Config\_list, 242  
sensor\_configuration.h, 243  
SensorManagement\_Sensor\_Config\_list, 244  
sensor\_ptr\_table  
SensorManagement, 125  
sensor\_type  
T\_SensorManagement\_Sensor\_Config, 144  
SensorManagement, 120  
getFullStringFormattedValue, 121  
getSensorCount, 122  
getSensorObjectPtr, 122  
nb\_sensors, 124  
sensor\_ptr\_table, 125  
SensorManagement, 121  
updateTaskPeriod, 124  
SensorManagement.cpp, 244  
p\_global\_ASW\_SensorManagement, 245  
SensorManagement.h, 245  
p\_global\_ASW\_SensorManagement, 246  
T\_SensorManagement\_Sensor\_Type, 246  
SensorManagement\_Sensor\_Config\_list  
sensor\_configuration.cpp, 242  
sensor\_configuration.h, 244  
sensorMgt\_ptr  
DebugManagement, 33  
setBaudRate  
uart, 155  
setBitRate  
I2C, 74  
SetDDRAMAddress  
LCD, 89  
setInfoStringPtr

DebugManagement, 30  
 setLastValidity  
     Sensor, 116  
 setLineAlignment  
     DisplayInterface, 59  
 setLineAlignmentAndRefresh  
     DisplayInterface, 60  
 setValidityTMO  
     Sensor, 117  
 shift\_data  
     T\_display\_data, 138  
 shiftLine\_task  
     DisplayInterface, 60  
 size  
     String, 134  
 startScheduling  
     scheduler, 109  
 startTimer1  
     timer, 152  
 stopTimer1  
     timer, 153  
 str\_cur\_ptr  
     T\_Display\_shift\_data, 139  
 str\_debug\_info\_message\_wdg\_disabled  
     DebugManagement.cpp, 181  
 str\_debug\_info\_message\_wdg\_enabled  
     DebugManagement.cpp, 182  
 str\_debug\_info\_message\_wdg\_tmo\_updated  
     DebugManagement.cpp, 182  
 str\_debug\_info\_message\_wdg\_tmo\_value  
     DebugManagement.cpp, 182  
 str\_debug\_info\_message\_wrong\_menu\_selection  
     DebugManagement.cpp, 182  
 str\_debug\_main\_menu  
     DebugManagement.cpp, 182  
 str\_debug\_wdg\_menu  
     DebugManagement.cpp, 183  
 str\_debug\_wdg\_timeout\_update\_selection  
     DebugManagement.cpp, 183  
 str\_ptr  
     T\_Display\_shift\_data, 139  
 String, 125  
     ~String, 127  
     appendBool, 128  
     appendChar, 128  
     appendInteger, 129  
     appendSpace, 130  
     appendString, 131  
     Clear, 132  
     ComputeStringSize, 132  
     getSize, 133  
     getString, 133  
     size, 134  
     String, 126, 127  
     string, 134  
 string  
     String, 134  
 String.cpp, 247  
 String.h, 247  
 SwitchWdg  
     Watchdog, 164  
 SystemReset  
     Watchdog, 165  
 systemReset  
     DebugManagement, 30  
 T\_ASW\_init\_cnf, 135  
     isDebugEnabled, 135  
     isDisplayActivated, 135  
     isLEDActivated, 135  
     isSensorMgtActivated, 136  
 T\_Display\_shift\_data, 138  
     str\_cur\_ptr, 139  
     str\_ptr, 139  
     temporization, 139  
 T\_DisplayInterface\_LineAlignment  
     DisplayInterface.h, 200  
 T\_DisplayInterface\_LineDisplayMode  
     DisplayInterface.h, 201  
 T\_LCD\_Command  
     LCD.h, 226  
 T\_LCD\_conf\_struct, 140  
     backlight\_en, 140  
     cursor\_en, 140  
     cursorBlink\_en, 140  
     display\_en, 141  
     entryModeDir, 141  
     entryModeShift, 141  
     fontType\_cnf, 141  
     i2c\_addr, 141  
     i2c\_bitrate, 142  
     lineNumber\_cnf, 142  
 T\_LCD\_config\_mode  
     LCD.h, 227  
 T\_LCD\_ram\_area  
     LCD.h, 227  
 T\_LL\_element  
     LinkedList, 96  
 T\_SensorManagement\_Sensor\_Config, 143  
     data\_name\_str, 144  
     period, 144  
     sensor\_type, 144  
     unit\_str, 144  
     validity\_tmo, 144  
 T\_SensorManagement\_Sensor\_Type  
     SensorManagement.h, 246  
 T\_display\_data, 136  
     alignment, 137  
     display\_str, 137  
     isEmpty, 137  
     mode, 137  
     shift\_data, 138  
 TASK\_PERIOD\_DEFAULT  
     Sensor.cpp, 240  
 TIMER\_CTC\_VALUE  
     scheduler.h, 238  
 task\_count

scheduler, 111  
task\_period  
    Sensor, 119  
Task\_t  
    scheduler, 104  
TaskPtr  
    scheduler::Task\_t, 145  
TaskPtr\_t  
    scheduler.h, 239  
TasksLL\_ptr  
    scheduler, 111  
TempSensor, 146  
    readTempSensor\_task, 148  
    TempSensor, 147  
    updateTaskPeriod, 149  
TempSensor.cpp, 248  
    DHT22\_PORT, 249  
TempSensor.h, 249  
temporization  
    T\_Display\_shift\_data, 139  
timeoutUpdate  
    Watchdog, 165  
timer, 150  
    configureTimer1, 151  
    getTimer1Value, 152  
    prescaler, 153  
    startTimer1, 152  
    stopTimer1, 153  
    timer, 151  
timer.cpp, 250  
    p\_global\_BSW\_timer, 250  
timer.h, 251  
    p\_global\_BSW\_timer, 251  
tmo\_value  
    Watchdog, 166  
USART\_BAUDRATE  
    DebugInterface.h, 179  
unit\_str  
    T\_SensorManagement\_Sensor\_Config, 144  
updateLineAndRefresh  
    DisplayInterface, 61  
updateTaskPeriod  
    HumSensor, 71  
    scheduler, 110  
    Sensor, 117  
    SensorManagement, 124  
    TempSensor, 149  
updateValidData  
    Sensor, 118  
usart, 154  
    BaudRate, 159  
    setBaudRate, 155  
    usart, 154  
    usart\_init, 155  
    usart\_read, 156  
    usart\_sendByte, 156  
    usart\_sendString, 157  
    usart\_transmit, 158  
    uart.cpp, 252  
        p\_global\_BSW\_usart, 252  
    usart.h, 253  
        p\_global\_BSW\_usart, 253  
    usart\_drv\_ptr  
        DebugInterface, 21  
    usart\_init  
        usart, 155  
    usart\_read  
        usart, 156  
    usart\_sendByte  
        usart, 156  
    usart\_sendString  
        usart, 157  
    usart\_transmit  
        usart, 158  
VALIDITY\_TIMEOUT\_MS\_DEFAULT  
    Sensor.cpp, 240  
valid\_pit  
    Sensor, 119  
valid\_value  
    Sensor, 119  
validity  
    Sensor, 119  
validity\_last\_read  
    Sensor, 119  
validity\_tmo  
    Sensor, 120  
    T\_SensorManagement\_Sensor\_Config, 144  
WDG\_TIMEOUT\_DEFAULT\_MS  
    Watchdog.cpp, 254  
WDG\_TMO\_120MS  
    Watchdog.h, 256  
WDG\_TMO\_15MS  
    Watchdog.h, 256  
WDG\_TMO\_1S  
    Watchdog.h, 257  
WDG\_TMO\_250MS  
    Watchdog.h, 257  
WDG\_TMO\_2S  
    Watchdog.h, 257  
WDG\_TMO\_30MS  
    Watchdog.h, 257  
WDG\_TMO\_4S  
    Watchdog.h, 257  
WDG\_TMO\_500MS  
    Watchdog.h, 258  
WDG\_TMO\_60MS  
    Watchdog.h, 258  
WDG\_TMO\_8S  
    Watchdog.h, 258  
Watchdog, 159  
    disable, 161  
    enable, 162  
    getTMOValue, 162  
    isActive, 166  
    isEnabled, 163

reset, 163  
SwitchWdg, 164  
SystemReset, 165  
timeoutUpdate, 165  
tmo\_value, 166  
Watchdog, 160  
Watchdog.cpp, 254  
  p\_global\_BSW\_wdg, 255  
  WDG\_TIMEOUT\_DEFAULT\_MS, 254  
Watchdog.h, 255  
  p\_global\_BSW\_wdg, 258  
  WDG\_TMO\_120MS, 256  
  WDG\_TMO\_15MS, 256  
  WDG\_TMO\_1S, 257  
  WDG\_TMO\_250MS, 257  
  WDG\_TMO\_2S, 257  
  WDG\_TMO\_30MS, 257  
  WDG\_TMO\_4S, 257  
  WDG\_TMO\_500MS, 258  
  WDG\_TMO\_60MS, 258  
  WDG\_TMO\_8S, 258  
WatchdogMenuManagement  
  DebugManagement, 31  
wdg\_state  
  debug\_mgt\_state\_struct\_t, 14  
welcomeMessageString  
  DisplayManagement.cpp, 203  
write  
  I2C, 74  
  LCD, 90  
write4bits  
  LCD, 91  
writeByte  
  I2C, 75  
WriteInRam  
  LCD, 92