

Arduino

1.0

Generated by Doxygen 1.8.14



# Contents

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

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

---

3.4.3.1	ClearLine()	24
3.4.3.2	DisplayFullLine()	25
3.4.3.3	FindFirstCharAddr()	26
3.4.3.4	getLLShiftDataPtr()	26
3.4.3.5	IsLineEmpty()	27
3.4.3.6	LLElementCompare()	27
3.4.3.7	shiftLine_task()	28
3.4.4	Member Data Documentation	29
3.4.4.1	dummy	29
3.4.4.2	lineEmptyTab	29
3.4.4.3	LL_shift_data_ptr	29
3.4.4.4	p_lcd	29
3.5	DisplayManagement Class Reference	30
3.5.1	Detailed Description	31
3.5.2	Constructor & Destructor Documentation	31
3.5.2.1	DisplayManagement()	31
3.5.3	Member Function Documentation	31
3.5.3.1	DisplaySensorData_Task()	32
3.5.3.2	GetIftPointer()	33
3.5.3.3	GetTempSensorPtr()	33
3.5.4	Member Data Documentation	34
3.5.4.1	p_display_ift	34
3.5.4.2	p_tempSensor	34
3.6	I2C Class Reference	34
3.6.1	Detailed Description	35
3.6.2	Constructor & Destructor Documentation	35
3.6.2.1	I2C()	35
3.6.3	Member Function Documentation	36
3.6.3.1	initializeBus()	36
3.6.3.2	setBitRate()	36

3.6.3.3	setTxAddress()	37
3.6.3.4	writeByte()	37
3.6.4	Member Data Documentation	38
3.6.4.1	bitrate	38
3.6.4.2	tx_address	38
3.7	keepAliveLed Class Reference	38
3.7.1	Detailed Description	39
3.7.2	Constructor & Destructor Documentation	39
3.7.2.1	keepAliveLed()	39
3.7.3	Member Function Documentation	39
3.7.3.1	blinkLed_task()	40
3.8	LCD Class Reference	40
3.8.1	Detailed Description	42
3.8.2	Constructor & Destructor Documentation	42
3.8.2.1	LCD()	42
3.8.3	Member Function Documentation	43
3.8.3.1	command()	43
3.8.3.2	ConfigureBacklight()	43
3.8.3.3	ConfigureCursorBlink()	44
3.8.3.4	ConfigureCursorOnOff()	45
3.8.3.5	ConfigureDisplayOnOff()	46
3.8.3.6	ConfigureEntryModeDir()	46
3.8.3.7	ConfigureEntryModeShift()	47
3.8.3.8	ConfigureFontType()	47
3.8.3.9	ConfigureI2CAddr()	48
3.8.3.10	ConfigureLineNumber()	49
3.8.3.11	GetDDRAMAddress()	50
3.8.3.12	GetLineNumberCnf()	50
3.8.3.13	InitializeScreen()	51
3.8.3.14	SetDDRAMAddress()	51

3.8.3.15	write()	52
3.8.3.16	write4bits()	53
3.8.3.17	WriteInRam()	54
3.8.4	Member Data Documentation	55
3.8.4.1	backlight_enable	55
3.8.4.2	cnfCursorBlink	55
3.8.4.3	cnfCursorOnOff	55
3.8.4.4	cnfDisplayOnOff	56
3.8.4.5	cnfEntryModeDir	56
3.8.4.6	cnfEntryModeShift	56
3.8.4.7	cnfFontType	56
3.8.4.8	cnfI2C_addr	56
3.8.4.9	cnfLineNumber	57
3.8.4.10	ddram_addr	57
3.9	LinkedList Class Reference	57
3.9.1	Detailed Description	58
3.9.2	Member Typedef Documentation	58
3.9.2.1	T_LL_element	58
3.9.3	Constructor & Destructor Documentation	59
3.9.3.1	LinkedList()	59
3.9.4	Member Function Documentation	59
3.9.4.1	AttachNewElement()	59
3.9.4.2	getCurrentElement()	60
3.9.4.3	IsLLEmpty()	60
3.9.4.4	MoveToNextElement()	61
3.9.4.5	RemoveElement()	61
3.9.4.6	ResetElementPtr()	62
3.9.5	Member Data Documentation	62
3.9.5.1	curElement_ptr	63
3.9.5.2	firstElement	63

3.10 scheduler Class Reference . . . . .	63
3.10.1 Detailed Description . . . . .	64
3.10.2 Member Typedef Documentation . . . . .	64
3.10.2.1 Task_t . . . . .	65
3.10.3 Constructor & Destructor Documentation . . . . .	65
3.10.3.1 scheduler() . . . . .	65
3.10.4 Member Function Documentation . . . . .	65
3.10.4.1 addPeriodicTask() . . . . .	65
3.10.4.2 getPitNumber() . . . . .	66
3.10.4.3 launchPeriodicTasks() . . . . .	67
3.10.4.4 LLElementCompare() . . . . .	68
3.10.4.5 removePeriodicTask() . . . . .	68
3.10.4.6 startScheduling() . . . . .	69
3.10.5 Member Data Documentation . . . . .	70
3.10.5.1 pit_number . . . . .	70
3.10.5.2 TasksLL_ptr . . . . .	70
3.11 String Class Reference . . . . .	70
3.11.1 Detailed Description . . . . .	71
3.11.2 Constructor & Destructor Documentation . . . . .	71
3.11.2.1 String() [1/2] . . . . .	71
3.11.2.2 String() [2/2] . . . . .	72
3.11.3 Member Function Documentation . . . . .	72
3.11.3.1 appendInteger() . . . . .	72
3.11.3.2 appendString() . . . . .	73
3.11.3.3 Clear() . . . . .	74
3.11.3.4 ComputeStringSize() . . . . .	75
3.11.3.5 getSize() . . . . .	76
3.11.3.6 getString() . . . . .	77
3.11.4 Member Data Documentation . . . . .	77
3.11.4.1 size . . . . .	77

3.11.4.2  string . . . . .	77
3.12  T_ASW_cnf_struct Struct Reference . . . . .	78
3.12.1  Detailed Description . . . . .	78
3.12.2  Member Data Documentation . . . . .	78
3.12.2.1  p_DisplayInterface . . . . .	79
3.12.2.2  p_DisplayManagement . . . . .	79
3.12.2.3  p_keepAliveLed . . . . .	79
3.12.2.4  p_TempSensor . . . . .	79
3.12.2.5  p_usartDebug . . . . .	79
3.13  T_BSW_cnf_struct Struct Reference . . . . .	80
3.13.1  Detailed Description . . . . .	80
3.13.2  Member Data Documentation . . . . .	80
3.13.2.1  p_cpupload . . . . .	80
3.13.2.2  p_dht22 . . . . .	81
3.13.2.3  p_dio . . . . .	81
3.13.2.4  p_i2c . . . . .	81
3.13.2.5  p_lcd . . . . .	81
3.13.2.6  p_timer . . . . .	81
3.13.2.7  p_usart . . . . .	82
3.14  T_Display_shift_data Struct Reference . . . . .	82
3.14.1  Detailed Description . . . . .	82
3.14.2  Member Data Documentation . . . . .	82
3.14.2.1  line . . . . .	82
3.14.2.2  size . . . . .	83
3.14.2.3  str_cur_ptr . . . . .	83
3.14.2.4  str_start_ptr . . . . .	83
3.14.2.5  temporization . . . . .	83
3.15  T_LCD_conf_struct Struct Reference . . . . .	83
3.15.1  Detailed Description . . . . .	84
3.15.2  Member Data Documentation . . . . .	84

---

3.15.2.1	backlight_en	84
3.15.2.2	cursor_en	84
3.15.2.3	cursorBlink_en	84
3.15.2.4	display_en	84
3.15.2.5	entryModeDir	84
3.15.2.6	entryModeShift	85
3.15.2.7	fontType_cnf	85
3.15.2.8	i2c_addr	85
3.15.2.9	lineNumber_cnf	85
3.16	LinkedList::T_LL_element Struct Reference	85
3.16.1	Detailed Description	86
3.16.2	Member Data Documentation	86
3.16.2.1	data_ptr	86
3.16.2.2	nextElement	86
3.17	scheduler::Task_t Struct Reference	86
3.17.1	Detailed Description	87
3.17.2	Member Data Documentation	87
3.17.2.1	period	87
3.17.2.2	TaskPtr	87
3.18	TempSensor Class Reference	87
3.18.1	Detailed Description	88
3.18.2	Constructor & Destructor Documentation	89
3.18.2.1	TempSensor()	89
3.18.3	Member Function Documentation	89
3.18.3.1	GetHumDecimal()	89
3.18.3.2	getHumidity()	90
3.18.3.3	GetHumInteger()	90
3.18.3.4	getHumPtr()	91
3.18.3.5	getTemp()	91
3.18.3.6	GetTempDecimal()	91

3.18.3.7	GetTempInteger()	92
3.18.3.8	getTempPtr()	93
3.18.3.9	GetValidity()	93
3.18.3.10	readTempSensor_task()	94
3.18.3.11	setValidity()	94
3.18.3.12	updateLastValidValues()	95
3.18.4	Member Data Documentation	95
3.18.4.1	read_humidity	96
3.18.4.2	read_temperature	96
3.18.4.3	valid_hum	96
3.18.4.4	valid_pit	96
3.18.4.5	valid_temp	96
3.18.4.6	validity	97
3.18.4.7	validity_last_read	97
3.19	timer Class Reference	97
3.19.1	Detailed Description	98
3.19.2	Constructor & Destructor Documentation	98
3.19.2.1	timer()	98
3.19.3	Member Function Documentation	98
3.19.3.1	configureTimer1()	98
3.19.3.2	getTimer1Value()	99
3.19.3.3	startTimer1()	100
3.19.3.4	stopTimer1()	100
3.19.4	Member Data Documentation	100
3.19.4.1	prescaler	100
3.20	uart Class Reference	101
3.20.1	Detailed Description	101
3.20.2	Constructor & Destructor Documentation	101
3.20.2.1	uart()	101
3.20.3	Member Function Documentation	102

---

3.20.3.1 <code>setBaudRate()</code>	102
3.20.3.2 <code>uart_init()</code>	103
3.20.3.3 <code>uart_read()</code>	103
3.20.3.4 <code>uart_sendString()</code>	104
3.20.3.5 <code>uart_transmit()</code>	104
3.20.4 Member Data Documentation	105
3.20.4.1 <code>BaudRate</code>	105
3.21 UsartDebug Class Reference	105
3.21.1 Detailed Description	106
3.21.2 Constructor & Destructor Documentation	106
3.21.2.1 <code>UsartDebug()</code>	106
3.21.3 Member Function Documentation	107
3.21.3.1 <code>activateDebugMode()</code>	107
3.21.3.2 <code>DebugModeManagement()</code>	108
3.21.3.3 <code>DisplayCPUload_task()</code>	109
3.21.3.4 <code>DisplaySensors_task()</code>	110
3.21.3.5 <code>isDebugModeActive()</code>	110
3.21.3.6 <code>sendBool()</code>	111
3.21.3.7 <code>sendInteger()</code>	112
3.21.3.8 <code>sendString()</code>	112
3.21.4 Member Data Documentation	113
3.21.4.1 <code>debug_state</code>	113
3.21.4.2 <code>debugModeActive_F</code>	113

<b>4 File Documentation</b>	<b>115</b>
4.1 asw.cpp File Reference . . . . .	115
4.1.1 Detailed Description . . . . .	116
4.1.2 Function Documentation . . . . .	116
4.1.2.1 asw_init() . . . . .	116
4.1.3 Variable Documentation . . . . .	116
4.1.3.1 ASW_cnf_struct . . . . .	117
4.2 asw.h File Reference . . . . .	117
4.2.1 Detailed Description . . . . .	117
4.2.2 Function Documentation . . . . .	118
4.2.2.1 asw_init() . . . . .	118
4.2.3 Variable Documentation . . . . .	118
4.2.3.1 ASW_cnf_struct . . . . .	118
4.3 bsw.cpp File Reference . . . . .	119
4.3.1 Detailed Description . . . . .	119
4.3.2 Function Documentation . . . . .	119
4.3.2.1 bsw_init() . . . . .	120
4.3.3 Variable Documentation . . . . .	120
4.3.3.1 BSW_cnf_struct . . . . .	120
4.4 bsw.h File Reference . . . . .	120
4.4.1 Detailed Description . . . . .	121
4.4.2 Macro Definition Documentation . . . . .	121
4.4.2.1 I2C_BITRATE . . . . .	121
4.4.2.2 USART_BAUDRATE . . . . .	122
4.4.3 Function Documentation . . . . .	122
4.4.3.1 bsw_init() . . . . .	122
4.4.4 Variable Documentation . . . . .	122
4.4.4.1 BSW_cnf_struct . . . . .	122
4.5 CpuLoad.cpp File Reference . . . . .	123
4.5.1 Detailed Description . . . . .	123

4.6 CpuLoad.h File Reference . . . . .	123
4.6.1 Detailed Description . . . . .	124
4.6.2 Macro Definition Documentation . . . . .	124
4.6.2.1 NB_OF_SAMPLES . . . . .	124
4.7 debug.cpp File Reference . . . . .	124
4.7.1 Detailed Description . . . . .	125
4.7.2 Variable Documentation . . . . .	125
4.7.2.1 str_debug_main_menu . . . . .	125
4.8 debug.h File Reference . . . . .	125
4.8.1 Detailed Description . . . . .	126
4.8.2 Macro Definition Documentation . . . . .	126
4.8.2.1 PERIOD_MS_TASK_DISPLAY_CPU_LOAD . . . . .	126
4.8.2.2 PERIOD_MS_TASK_DISPLAY_SENSORS . . . . .	126
4.8.3 Enumeration Type Documentation . . . . .	126
4.8.3.1 debug_state_t . . . . .	126
4.9 dht22.cpp File Reference . . . . .	127
4.9.1 Detailed Description . . . . .	127
4.9.2 Macro Definition Documentation . . . . .	128
4.9.2.1 MAX_WAIT_TIME_US . . . . .	128
4.10 dht22.h File Reference . . . . .	128
4.10.1 Detailed Description . . . . .	128
4.10.2 Macro Definition Documentation . . . . .	129
4.10.2.1 DHT22_PORT . . . . .	129
4.11 dio.cpp File Reference . . . . .	129
4.11.1 Detailed Description . . . . .	129
4.12 dio.h File Reference . . . . .	130
4.12.1 Detailed Description . . . . .	130
4.12.2 Macro Definition Documentation . . . . .	130
4.12.2.1 DECODE_PIN . . . . .	131
4.12.2.2 DECODE_PORT . . . . .	131

---

4.12.2.3 ENCODE_PORT . . . . .	131
4.12.2.4 PORT_A . . . . .	131
4.12.2.5 PORT_B . . . . .	131
4.12.2.6 PORT_C . . . . .	132
4.12.2.7 PORT_CNF_IN . . . . .	132
4.12.2.8 PORT_CNF_OUT . . . . .	132
4.12.2.9 PORT_D . . . . .	132
4.13 dio_port_cnf.h File Reference . . . . .	132
4.13.1 Detailed Description . . . . .	133
4.13.2 Macro Definition Documentation . . . . .	133
4.13.2.1 PORTB_CNF_DDRB . . . . .	133
4.13.2.2 PORTB_CNF_PORTB . . . . .	133
4.14 dio_reg_atm2560.h File Reference . . . . .	134
4.14.1 Macro Definition Documentation . . . . .	134
4.14.1.1 DDRA_PTR . . . . .	134
4.14.1.2 DDRB_PTR . . . . .	134
4.14.1.3 DDRC_PTR . . . . .	134
4.14.1.4 DDRD_PTR . . . . .	135
4.14.1.5 PINA_PTR . . . . .	135
4.14.1.6 PINB_PTR . . . . .	135
4.14.1.7 PINC_PTR . . . . .	135
4.14.1.8 PIND_PTR . . . . .	135
4.14.1.9 PORTA_PTR . . . . .	136
4.14.1.10 PORTB_PTR . . . . .	136
4.14.1.11 PORTC_PTR . . . . .	136
4.14.1.12 PORTD_PTR . . . . .	136
4.15 DisplayInterface.cpp File Reference . . . . .	137
4.15.1 Detailed Description . . . . .	137
4.16 DisplayInterface.h File Reference . . . . .	137
4.16.1 Detailed Description . . . . .	138

4.16.2 Macro Definition Documentation . . . . .	138
4.16.2.1 DISPLAY_LINE_SHIFT_PERIOD_MS . . . . .	138
4.16.2.2 DISPLAY_LINE_SHIFT_TEMPO_TIME . . . . .	138
4.16.3 Enumeration Type Documentation . . . . .	139
4.16.3.1 T_DisplayInterface_LineDisplayMode . . . . .	139
4.17 DisplayManagement.cpp File Reference . . . . .	139
4.17.1 Detailed Description . . . . .	139
4.18 DisplayManagement.h File Reference . . . . .	140
4.18.1 Detailed Description . . . . .	141
4.18.2 Macro Definition Documentation . . . . .	141
4.18.2.1 DISPLAY_MGT_LCD_I2C_ADDR . . . . .	141
4.18.2.2 DISPLAY_MGT_LINE_HUM . . . . .	141
4.18.2.3 DISPLAY_MGT_LINE_TEMP . . . . .	141
4.18.2.4 DISPLAY_MGT_PERIOD_TASK_SENSOR . . . . .	141
4.18.3 Variable Documentation . . . . .	142
4.18.3.1 humidityDisplayString . . . . .	142
4.18.3.2 LCD_init_cnf . . . . .	142
4.18.3.3 tempDisplayString . . . . .	142
4.19 I2C.cpp File Reference . . . . .	143
4.19.1 Detailed Description . . . . .	143
4.20 I2C.h File Reference . . . . .	143
4.20.1 Detailed Description . . . . .	144
4.20.2 Macro Definition Documentation . . . . .	144
4.20.2.1 DATA_ACK . . . . .	144
4.20.2.2 SLA_ACK . . . . .	144
4.20.2.3 START . . . . .	145
4.21 keepAliveLed.cpp File Reference . . . . .	145
4.21.1 Detailed Description . . . . .	145
4.22 keepAliveLed.h File Reference . . . . .	145
4.22.1 Detailed Description . . . . .	146

---

4.22.2 Macro Definition Documentation . . . . .	146
4.22.2.1 LED_PORT . . . . .	146
4.22.2.2 PERIOD_MS_TASK_LED . . . . .	146
4.23 LCD.cpp File Reference . . . . .	147
4.23.1 Detailed Description . . . . .	147
4.24 LCD.h File Reference . . . . .	147
4.24.1 Detailed Description . . . . .	149
4.24.2 Macro Definition Documentation . . . . .	149
4.24.2.1 BACKLIGHT_PIN . . . . .	149
4.24.2.2 EN_PIN . . . . .	149
4.24.2.3 LCD_CNF_BACKLIGHT_OFF . . . . .	150
4.24.2.4 LCD_CNF_BACKLIGHT_ON . . . . .	150
4.24.2.5 LCD_CNF_CURSOR_BLINK_OFF . . . . .	150
4.24.2.6 LCD_CNF_CURSOR_BLINK_ON . . . . .	150
4.24.2.7 LCD_CNF_CURSOR_OFF . . . . .	150
4.24.2.8 LCD_CNF_CURSOR_ON . . . . .	151
4.24.2.9 LCD_CNF_DISPLAY_OFF . . . . .	151
4.24.2.10 LCD_CNF_DISPLAY_ON . . . . .	151
4.24.2.11 LCD_CNF_ENTRY_MODE_DIRECTION_LEFT . . . . .	151
4.24.2.12 LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT . . . . .	151
4.24.2.13 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF . . . . .	152
4.24.2.14 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON . . . . .	152
4.24.2.15 LCD_CNF_FONT_5_11 . . . . .	152
4.24.2.16 LCD_CNF_FONT_5_8 . . . . .	152
4.24.2.17 LCD_CNF_ONE_LINE . . . . .	152
4.24.2.18 LCD_CNF_SHIFT_ID . . . . .	153
4.24.2.19 LCD_CNF_SHIFT_SH . . . . .	153
4.24.2.20 LCD_CNF_TWO_LINE . . . . .	153
4.24.2.21 LCD_DISPLAY_CTRL_FIELD_B . . . . .	153
4.24.2.22 LCD_DISPLAY_CTRL_FIELD_C . . . . .	153

4.24.2.23 LCD_DISPLAY_CTRL_FIELD_D . . . . .	154
4.24.2.24 LCD_FCT_SET_FIELD_DL . . . . .	154
4.24.2.25 LCD_FCT_SET_FIELD_F . . . . .	154
4.24.2.26 LCD_FCT_SET_FIELD_N . . . . .	154
4.24.2.27 LCD_INST_CLR_DISPLAY_BIT . . . . .	154
4.24.2.28 LCD_INST_DISPLAY_CTRL . . . . .	155
4.24.2.29 LCD_INST_ENTRY_MODE_SET . . . . .	155
4.24.2.30 LCD_INST_FUNCTION_SET . . . . .	155
4.24.2.31 LCD_INST_SET_DDRAM_ADDR . . . . .	155
4.24.2.32 LCD_RAM_1_LINE_MAX . . . . .	155
4.24.2.33 LCD_RAM_1_LINE_MIN . . . . .	156
4.24.2.34 LCD_RAM_2_LINES_MAX_1 . . . . .	156
4.24.2.35 LCD_RAM_2_LINES_MAX_2 . . . . .	156
4.24.2.36 LCD_RAM_2_LINES_MIN_1 . . . . .	156
4.24.2.37 LCD_RAM_2_LINES_MIN_2 . . . . .	156
4.24.2.38 LCD_SIZE_NB_CHAR_PER_LINE . . . . .	157
4.24.2.39 LCD_SIZE_NB_LINES . . . . .	157
4.24.2.40 LCD_WAIT_CLR_RETURN . . . . .	157
4.24.2.41 LCD_WAIT_OTHER_MODES . . . . .	157
4.24.2.42 RS_PIN . . . . .	157
4.24.2.43 RW_PIN . . . . .	158
4.24.3 Enumeration Type Documentation . . . . .	158
4.24.3.1 T_LCD_command . . . . .	158
4.24.3.2 T_LCD_config_mode . . . . .	158
4.24.3.3 T_LCD_ram_area . . . . .	159
4.25 LinkedList.cpp File Reference . . . . .	159
4.26 LinkedList.h File Reference . . . . .	159
4.26.1 Detailed Description . . . . .	160
4.26.2 Typedef Documentation . . . . .	160
4.26.2.1 CompareFctPtr_t . . . . .	160

---

4.27 main.cpp File Reference . . . . .	160
4.27.1 Detailed Description . . . . .	161
4.27.2 Function Documentation . . . . .	161
4.27.2.1 ISR() [1/2] . . . . .	161
4.27.2.2 ISR() [2/2] . . . . .	162
4.27.2.3 main() . . . . .	162
4.28 main.h File Reference . . . . .	163
4.28.1 Detailed Description . . . . .	163
4.29 operators.cpp File Reference . . . . .	163
4.29.1 Detailed Description . . . . .	164
4.29.2 Function Documentation . . . . .	164
4.29.2.1 operator delete() . . . . .	164
4.29.2.2 operator new() . . . . .	164
4.30 operators.h File Reference . . . . .	166
4.30.1 Detailed Description . . . . .	166
4.30.2 Function Documentation . . . . .	167
4.30.2.1 operator delete() . . . . .	167
4.30.2.2 operator new() . . . . .	167
4.31 scheduler.cpp File Reference . . . . .	167
4.31.1 Detailed Description . . . . .	168
4.31.2 Variable Documentation . . . . .	168
4.31.2.1 p_scheduler . . . . .	168
4.32 scheduler.h File Reference . . . . .	169
4.32.1 Detailed Description . . . . .	169
4.32.2 Macro Definition Documentation . . . . .	170
4.32.2.1 PRESCALER_PERIODIC_TIMER . . . . .	170
4.32.2.2 SW_PERIOD_MS . . . . .	170
4.32.2.3 TIMER_CTC_VALUE . . . . .	170
4.32.3 Typedef Documentation . . . . .	170
4.32.3.1 TaskPtr_t . . . . .	170

4.32.4 Variable Documentation . . . . .	170
4.32.4.1 p_scheduler . . . . .	171
4.33 String.cpp File Reference . . . . .	171
4.33.1 Detailed Description . . . . .	171
4.34 String.h File Reference . . . . .	172
4.34.1 Detailed Description . . . . .	172
4.35 TempSensor.cpp File Reference . . . . .	172
4.35.1 Detailed Description . . . . .	173
4.35.2 Macro Definition Documentation . . . . .	173
4.35.2.1 PIT_BEFORE_INVALID . . . . .	173
4.36 TempSensor.h File Reference . . . . .	174
4.36.1 Detailed Description . . . . .	174
4.36.2 Macro Definition Documentation . . . . .	174
4.36.2.1 PERIOD_MS_TASK_TEMP_SENSOR . . . . .	174
4.37 timer.cpp File Reference . . . . .	175
4.37.1 Detailed Description . . . . .	175
4.38 timer.h File Reference . . . . .	175
4.38.1 Detailed Description . . . . .	176
4.39 usart.cpp File Reference . . . . .	176
4.39.1 Detailed Description . . . . .	176
4.40 usart.h File Reference . . . . .	177
4.40.1 Detailed Description . . . . .	177
Index . . . . .	179

# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">CpuLoad</a>	Class defining CPU load libraries . . . . .	5
<a href="#">dht22</a>	DHT 22 driver class . . . . .	10
<a href="#">dio</a>	DIO class . . . . .	12
<a href="#">DisplayInterface</a>	Display interface services class . . . . .	22
<a href="#">DisplayManagement</a>	Display management class . . . . .	30
<a href="#">I2C</a>	Two-wire serial interface ( <a href="#">I2C</a> ) class definition . . . . .	34
<a href="#">keepAliveLed</a>	Class for keep-alive LED blinking . . . . .	38
<a href="#">LCD</a>	Class for <a href="#">LCD</a> S2004A display driver . . . . .	40
<a href="#">LinkedList</a>	Linked list class . . . . .	57
<a href="#">scheduler</a>	Scheduler class . . . . .	63
<a href="#">String</a>	<a href="#">String</a> management class . . . . .	70
<a href="#">T_ASW_cnf_struct</a>	ASW configuration structure . . . . .	78
<a href="#">T_BSW_cnf_struct</a>	BSW configuration structure . . . . .	80
<a href="#">T_Display_shift_data</a>	Structure containing shift data . . . . .	82
<a href="#">T_LCD_conf_struct</a>	. . . . .	83
<a href="#">LinkedList::T_LL_element</a>	Type defining a linked list element . . . . .	85
<a href="#">scheduler::Task_t</a>	Type defining a task structure . . . . .	86
<a href="#">TempSensor</a>	Class for temperature sensor . . . . .	87

<a href="#">timer</a>	Class defining a timer . . . . .	97
<a href="#">usart</a>	USART serial bus class . . . . .	101
<a href="#">UsartDebug</a>	Class used for debugging on usart link . . . . .	105

# Chapter 2

## File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">asw.cpp</a>	ASW main file . . . . .	115
<a href="#">asw.h</a>	ASW main header file . . . . .	117
<a href="#">bsw.cpp</a>	BSW main file . . . . .	119
<a href="#">bsw.h</a>	BSW main header file . . . . .	120
<a href="#">CpuLoad.cpp</a>	Defines functions of class <a href="#">CpuLoad</a> . . . . .	123
<a href="#">CpuLoad.h</a>	<a href="#">CpuLoad</a> class header file . . . . .	123
<a href="#">debug.cpp</a>	This file defines classes for log and debug data transmission on USART link . . . . .	124
<a href="#">debug.h</a>	Header file for debug and logging functions . . . . .	125
<a href="#">dht22.cpp</a>	This file defines classes for DHT22 driver . . . . .	127
<a href="#">dht22.h</a>	DHT22 driver header file . . . . .	128
<a href="#">dio.cpp</a>	DIO library . . . . .	129
<a href="#">dio.h</a>	DIO library header file . . . . .	130
<a href="#">dio_port_cnf.h</a>	Digital ports configuration file . . . . .	132
<a href="#">dio_reg_atm2560.h</a>		134
<a href="#">DisplayInterface.cpp</a>	Source code file for display services . . . . .	137
<a href="#">DisplayInterface.h</a>	<a href="#">DisplayInterface</a> class header file . . . . .	137
<a href="#">DisplayManagement.cpp</a>	Display management source file . . . . .	139
<a href="#">DisplayManagement.h</a>	Display management class header file . . . . .	140

I2C.cpp	Two-wire interface ( <a href="#">I2C</a> ) source file . . . . .	143
I2C.h	<a href="#">I2C</a> class header file . . . . .	143
keepAliveLed.cpp	Definition of function for class <a href="#">keepAliveLed</a> . . . . .	145
keepAliveLed.h	Class <a href="#">keepAliveLed</a> header file . . . . .	145
LCD.cpp	<a href="#">LCD</a> class source file . . . . .	147
LCD.h	<a href="#">LCD</a> class header file . . . . .	147
LinkedList.cpp		159
LinkedList.h	Linked List library header file . . . . .	159
main.cpp	Background task file . . . . .	160
main.h	Background task header file . . . . .	163
operators.cpp	C++ operators definitions . . . . .	163
operators.h	C++ operators definitions header file . . . . .	166
scheduler.cpp	Defines scheduler class . . . . .	167
scheduler.h	Scheduler class header file . . . . .	169
String.cpp	<a href="#">String</a> class source file . . . . .	171
String.h	<a href="#">String</a> class header file . . . . .	172
TempSensor.cpp	Defines function of class <a href="#">TempSensor</a> . . . . .	172
TempSensor.h	Class <a href="#">TempSensor</a> header file . . . . .	174
timer.cpp	Defines function for class timer . . . . .	175
timer.h	Timer class header file . . . . .	175
uart.cpp	BSW library for USART . . . . .	176
uart.h	Header file for USART library . . . . .	177

# Chapter 3

## Class Documentation

### 3.1 CpuLoad Class Reference

Class defining CPU load libraries.

```
#include <CpuLoad.h>
```

#### Public Member Functions

- [CpuLoad \(\)](#)  
*CpuLoad class constructor.*
- [void ComputeCPULoad \(\)](#)  
*Computes current CPU load.*
- [uint8\\_t getCurrentCPULoad \(\)](#)  
*Get current CPU load value.*
- [uint8\\_t getAverageCPULoad \(\)](#)  
*Get average CPU load value.*
- [uint8\\_t getMaxCPULoad \(\)](#)  
*Get maximum CPU load value.*

#### Private Attributes

- [uint8\\_t current\\_load](#)
- [uint8\\_t avg\\_load](#)
- [uint8\\_t max\\_load](#)
- [uint8\\_t sample\\_cnt](#)
- [uint8\\_t sample\\_mem \[NB\\_OF\\_SAMPLES\]](#)
- [uint8\\_t sample\\_idx](#)
- [uint16\\_t last\\_sum\\_value](#)

#### 3.1.1 Detailed Description

Class defining CPU load libraries.

This class defines tools to compute and monitor CPU load.

Definition at line 19 of file CpuLoad.h.

### 3.1.2 Constructor & Destructor Documentation

#### 3.1.2.1 CpuLoad()

`CpuLoad::CpuLoad ( )`

`CpuLoad` class constructor.

This function initializes class `CpuLoad`

Returns

Nothing

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

### 3.1.3 Member Function Documentation

#### 3.1.3.1 ComputeCPULoad()

`void CpuLoad::ComputeCPULoad ( )`

Computes current CPU load.

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

Returns

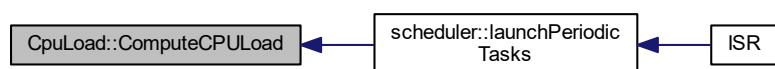
Nothing

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

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.1.3.2 getAverageCPULoad()

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

Get average CPU load value.

This function returns the average CPU load value

#### Returns

Average CPU load value

Definition at line 56 of file CpuLoad.h.

Here is the caller graph for this function:



### 3.1.3.3 getCurrentCPULoad()

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

Get current CPU load value.

This function returns the current CPU load value

#### Returns

Current CPU load value

Definition at line 45 of file CpuLoad.h.

Here is the caller graph for this function:



### 3.1.3.4 getMaxCPUload()

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

Get maximum CPU load value.

This function returns the maximum CPU load value

#### Returns

Maximum CPU load value

Definition at line 67 of file CpuLoad.h.

Here is the caller graph for this function:



### 3.1.4 Member Data Documentation

#### 3.1.4.1 avg\_load

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

Average CPU load based on the last 50 cycles

Definition at line 74 of file CpuLoad.h.

#### 3.1.4.2 current\_load

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

Current CPU load (load of last cycle)

Definition at line 73 of file CpuLoad.h.

#### 3.1.4.3 last\_sum\_value

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

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

Definition at line 79 of file CpuLoad.h.

#### 3.1.4.4 max\_load

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

Maximum CPU load since power on

Definition at line 75 of file CpuLoad.h.

#### 3.1.4.5 sample\_cnt

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

Number of samples used to compute average load

Definition at line 76 of file CpuLoad.h.

#### 3.1.4.6 sample\_idx

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

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

Definition at line 78 of file CpuLoad.h.

#### 3.1.4.7 sample\_mem

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

Memorization of the last NB\_OF\_SAMPLES measures

Definition at line 77 of file CpuLoad.h.

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

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

## 3.2 dht22 Class Reference

DHT 22 driver class.

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

### Public Member Functions

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

### Private Member Functions

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

#### 3.2.1 Detailed Description

DHT 22 driver class.

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

Definition at line 22 of file dht22.h.

#### 3.2.2 Constructor & Destructor Documentation

##### 3.2.2.1 dht22()

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

[dht22](#) class constructor

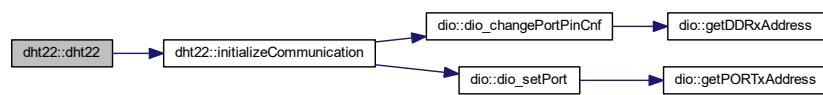
Initializes the class [dht22](#)

Returns

Nothing

Definition at line 22 of file dht22.cpp.

Here is the call graph for this function:



### 3.2.3 Member Function Documentation

#### 3.2.3.1 initializeCommunication()

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

Initializes the communication.

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

##### Returns

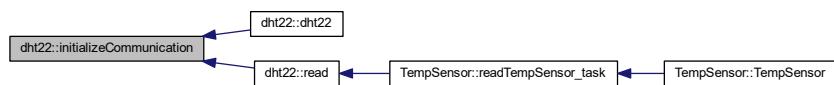
Nothing

Definition at line 198 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 3.2.3.2 read()

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

Reads the data from DHT22.

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

### Parameters

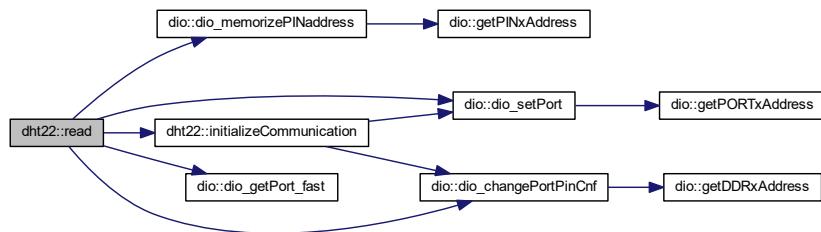
<code>out</code>	<code>raw_humidity</code>	Raw humidity value received from sensor
<code>out</code>	<code>raw_temperature</code>	Raw temperature value received from sensor

### Returns

Validity of the read value

Definition at line 27 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

## 3.3 dio Class Reference

DIO class.

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

## Public Member Functions

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

## Private Member Functions

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

## Private Attributes

- `uint8_t * PINx_addr_mem`
- `uint8_t PINx_idx_mem`

### 3.3.1 Detailed Description

DIO class.

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

Definition at line 31 of file dio.h.

### 3.3.2 Constructor & Destructor Documentation

### 3.3.2.1 dio()

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

dio class constructor

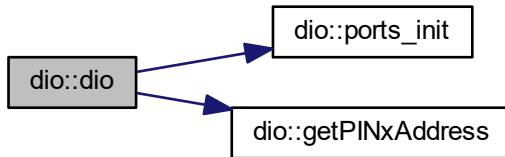
Initializes class dio and calls DIO hardware initialization function

Returns

Nothing

Definition at line 112 of file dio.cpp.

Here is the call graph for this function:



## 3.3.3 Member Function Documentation

### 3.3.3.1 dio\_changePortPinCnf()

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

Changes the IO configuration of the selected pin.

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

Parameters

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

**Returns**

Nothing

Definition at line 149 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.3.3.2 dio\_getPort()**

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

Gets the logical state of selected pin.

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

**Parameters**

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

**Returns**

Logical state of selected pin

Definition at line 139 of file dio.cpp.

Here is the call graph for this function:



### 3.3.3.3 dio\_getPort\_fast()

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

Gets the logical state of the memorized pin.

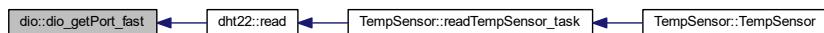
This function gets the logical value of the memorized pin. The corresponding port and pin index are stored in class members PINx\_addr\_mem and PINx\_idx\_mem. This mechanism is used to speed up reading time as this function no longer needs to extract register address and pin index from portcode.

#### Returns

Logical state of selected pin

Definition at line 171 of file dio.cpp.

Here is the caller graph for this function:



### 3.3.3.4 dio\_invertPort()

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

Inverts the state of output port.

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

**Parameters**

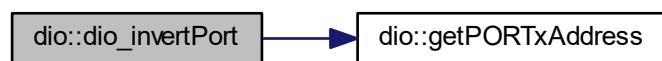
in	<i>portcode</i>	Encoded pin and register index
----	-----------------	--------------------------------

**Returns**

Nothing

Definition at line 131 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.3.3.5 dio\_memorizePINaddress()**

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

Memorizes PINx register address and pin index.

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

**Parameters**

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

**Returns**

Nothing

Definition at line 165 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.3.3.6 dio\_setPort()**

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

Port setting function.

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

**Parameters**

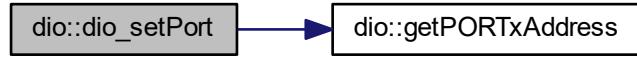
in	<i>portcode</i>	Encoded pin and register index
in	<i>state</i>	Requested state to set pin

**Returns**

Nothing

Definition at line 121 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.3.3.7 getDDRxAddress()

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

Gets the physical address of the requested register DDRx.

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

#### Parameters

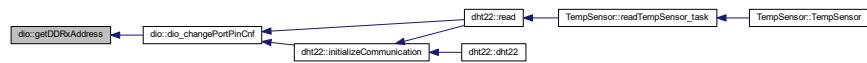
in	<i>portcode</i>	Encoded port code
----	-----------------	-------------------

#### Returns

Pointer to the DDRx register

Definition at line 83 of file dio.cpp.

Here is the caller graph for this function:



### 3.3.3.8 getPINxAddress()

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

Gets the physical address of the requested register PINx.

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

#### Parameters

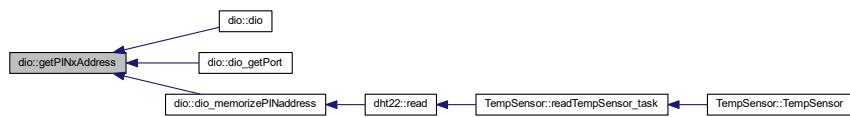
in	<i>portcode</i>	Encoded port code
----	-----------------	-------------------

#### Returns

Pointer to the PINx register

Definition at line 54 of file dio.cpp.

Here is the caller graph for this function:



### 3.3.3.9 getPORTxAddress()

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

Gets the physical address of the requested register PORTx.

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

#### Parameters

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

#### Returns

Pointer to the PORTx register

Definition at line 25 of file dio.cpp.

Here is the caller graph for this function:



### 3.3.3.10 ports\_init()

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

Digital ports hardware initialization function.

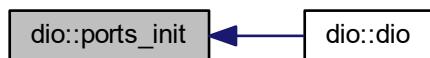
This function initializes digital ports as input or output and sets their initial values

**Returns**

Nothing

Definition at line 16 of file dio.cpp.

Here is the caller graph for this function:



### 3.3.4 Member Data Documentation

#### 3.3.4.1 PINx\_addr\_mem

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

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

Definition at line 146 of file dio.h.

### 3.3.4.2 PINx\_idx\_mem

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

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

Definition at line 147 of file dio.h.

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

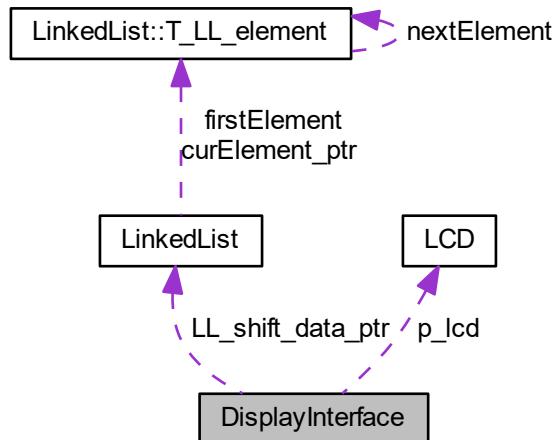
- [dio.h](#)
- [dio.cpp](#)

## 3.4 DisplayInterface Class Reference

Display interface services class.

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

Collaboration diagram for DisplayInterface:



### Public Member Functions

- [DisplayInterface \(const T\\_LCD\\_conf\\_struct \\*LCD\\_init\\_cnf\)](#)  
*Class constructor.*
- [bool DisplayFullLine \(uint8\\_t \\*str, uint8\\_t size, uint8\\_t line, T\\_DisplayInterface\\_LineDisplayMode mode\)](#)  
*Line display function.*
- [bool ClearLine \(uint8\\_t line\)](#)  
*Line clearing function.*
- [bool IsLineEmpty \(uint8\\_t line\)](#)  
*Empty line get function.*
- [LinkedList \\* getLLShiftDataPtr \(\)](#)  
*Linked list shift data get function.*

## Static Public Member Functions

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

## Private Member Functions

- uint8\_t [FindFirstCharAddr](#) (uint8\_t line)  
*Finds start address of a line.*

## Private Attributes

- [LCD \\* p\\_lcd](#)
- uint32\_t [dummy](#)
- bool [lineEmptyTab \[LCD\\_SIZE\\_NB\\_LINES\]](#)
- [LinkedList \\* LL\\_shift\\_data\\_ptr](#)

### 3.4.1 Detailed Description

Display interface services class.

This class defines the services used for interfacing display management services and [LCD](#) screen driver

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

### 3.4.2 Constructor & Destructor Documentation

#### 3.4.2.1 [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	<a href="#">LCD_init_cnf</a>	Initial configuration of the screen
----	------------------------------	-------------------------------------

## Returns

Nothing

Definition at line 37 of file DisplayInterface.cpp.

### 3.4.3 Member Function Documentation

### 3.4.3.1 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

## Parameters

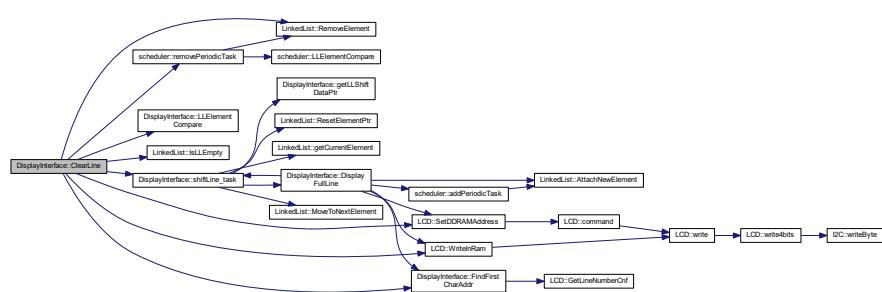
in	<i>line</i>	Line to clear
----	-------------	---------------

## Returns

True if the line has been cleared, false otherwise

Definition at line 167 of file DisplayInterface.cpp.

Here is the call graph for this function:



### 3.4.3.2 DisplayFullLine()

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

Line display function.

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

#### Parameters

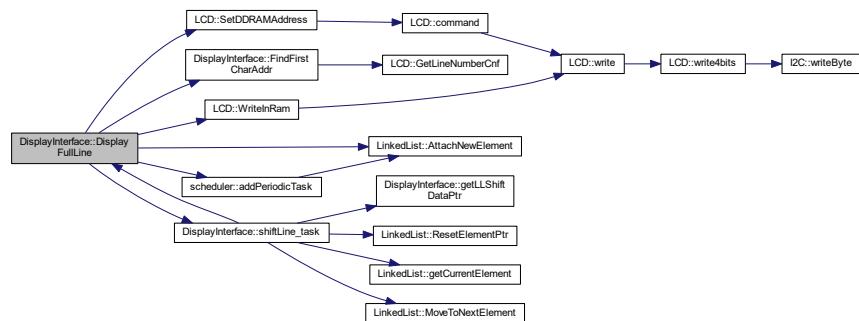
in	<i>str</i>	Pointer to the string to display
in	<i>size</i>	Size of the string to display
in	<i>line</i>	Index of the line where the string shall be displayed
in	<i>mode</i>	Display mode

#### Returns

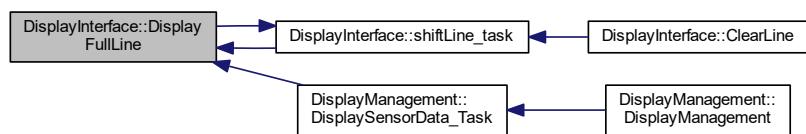
True if the line has been correctly displayed, false otherwise

Definition at line 59 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.4.3.3 FindFirstCharAddr()

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

Finds start address of a line.

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

#### Parameters

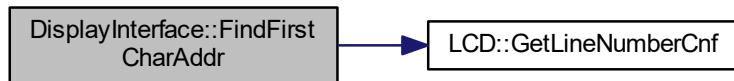
<code>in</code>	<code>line</code>	Line which address shall be found
-----------------	-------------------	-----------------------------------

#### Returns

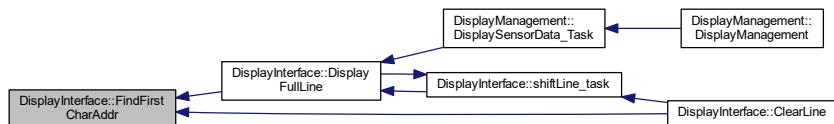
Address in DDRAM of the first character of the line

Definition at line 128 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.4.3.4 getLLShiftDataPtr()

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

Linked list shift data get function.

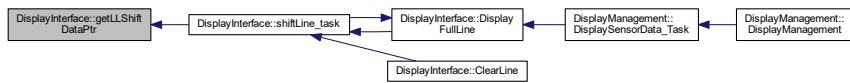
This function returns the pointer to the shift data linked list.

**Returns**

Pointer to linked list

Definition at line 119 of file `DisplayInterface.h`.

Here is the caller graph for this function:

**3.4.3.5 IsLineEmpty()**

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

Empty line get function.

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

**Parameters**

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

**Returns**

True if the line is empty, false otherwise

Definition at line 198 of file `DisplayInterface.cpp`.

**3.4.3.6 LLElementCompare()**

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

Linked list comparison function.

This function is called by the linked list class to compare one element of the list to a given element. In the class `DisplayInterface`, the `LLElement` is a shift data pointer (containing a line number inside it), and the `compareElement` a line number. The comparison will be done between the two function pointer.

### Parameters

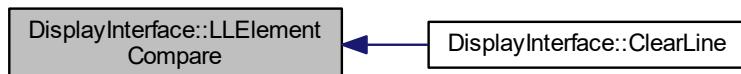
in	<i>LLElement</i>	Pointer to the linked list element
in	<i>CompareElement</i>	Pointer to the element to the compare

### Returns

True if both elements are identical, false otherwise

Definition at line 207 of file DisplayInterface.cpp.

Here is the caller graph for this function:



### 3.4.3.7 shiftLine\_task()

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

Line shifting periodic task.

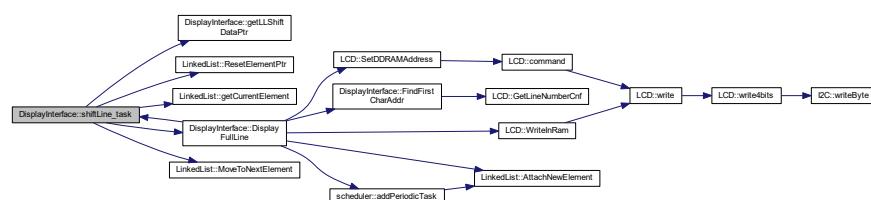
This function is called periodically by the scheduler. It shifts all the lines in line shifting mode and updates the data structures.

### Returns

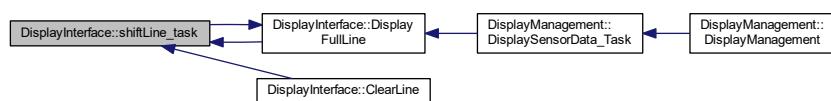
Nothing

Definition at line 218 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.4.4 Member Data Documentation

#### 3.4.4.1 dummy

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

Needed for data alignment

Definition at line 127 of file DisplayInterface.h.

#### 3.4.4.2 lineEmptyTab

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

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

Definition at line 128 of file DisplayInterface.h.

#### 3.4.4.3 LL\_shift\_data\_ptr

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

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

Definition at line 129 of file DisplayInterface.h.

#### 3.4.4.4 p\_lcd

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

Pointer to the attached LCD driver object

Definition at line 126 of file DisplayInterface.h.

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

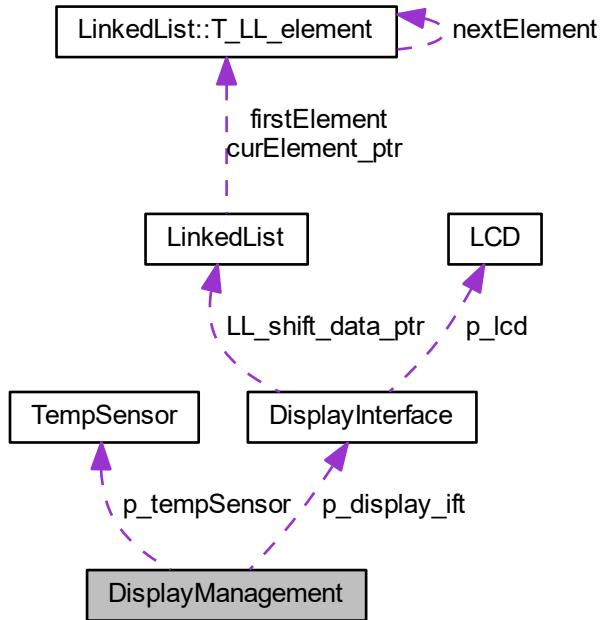
- [DisplayInterface.h](#)
- [DisplayInterface.cpp](#)

### 3.5 DisplayManagement Class Reference

Display management class.

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

Collaboration diagram for DisplayManagement:



#### Public Member Functions

- `DisplayManagement ()`  
*Class constructor.*
- `DisplayInterface * GetIfpPointer ()`  
*Interface pointer get function.*
- `TempSensor * GetTempSensorPtr ()`  
*Sensor pointer get function.*

#### Static Public Member Functions

- `static void DisplaySensorData_Task ()`  
*Periodic task for displaying sensor data.*

#### Private Attributes

- `DisplayInterface * p_display_ift`
- `TempSensor * p_tempSensor`

### 3.5.1 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 45 of file DisplayManagement.h.

### 3.5.2 Constructor & Destructor Documentation

### 3.5.2.1 DisplayManagement()

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

## Class constructor.

This class initializes display management.

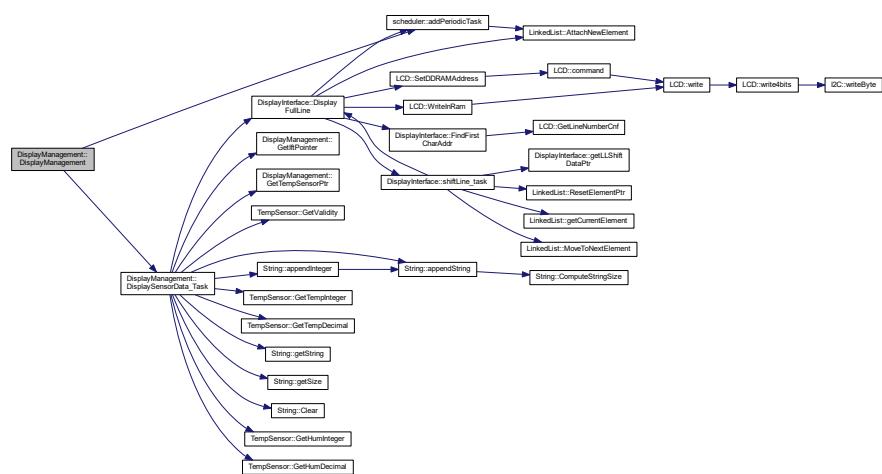
It created a display interface object and initializes all class variables.

## Returns

Nothing

Definition at line 30 of file DisplayManagement.cpp.

Here is the call graph for this function:



### 3.5.3 Member Function Documentation

### 3.5.3.1 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.

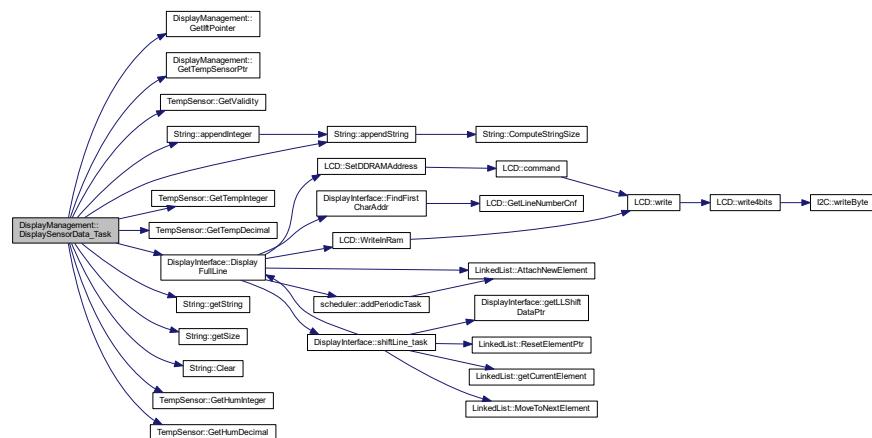
It is called periodically by scheduler.

#### Returns

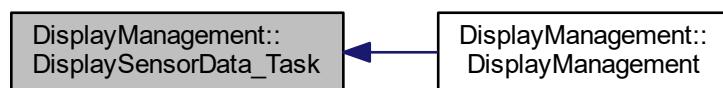
Nothing

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

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.5.3.2 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 74 of file DisplayManagement.h.

Here is the caller graph for this function:



### 3.5.3.3 GetTempSensorPtr()

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

Sensor pointer get function.

This function returns the pointer to the temperature sensor object

#### Returns

Pointer to sensor object

Definition at line 85 of file DisplayManagement.h.

Here is the caller graph for this function:



### 3.5.4 Member Data Documentation

#### 3.5.4.1 p\_display\_ift

`DisplayInterface* DisplayManagement::p_display_ift [private]`

Pointer to the display interface object

Definition at line 92 of file `DisplayManagement.h`.

#### 3.5.4.2 p\_tempSensor

`TempSensor* DisplayManagement::p_tempSensor [private]`

Pointer to the temperature sensor object

Definition at line 93 of file `DisplayManagement.h`.

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

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

## 3.6 I2C Class Reference

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

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

### Public Member Functions

- [`I2C` \(uint32\\_t l\\_bitrate\)](#)  
*I2C class constructor.*
- [`bool writeByte \(uint8\_t \*data\)`](#)  
*Byte sending function.*
- [`void setTxAddress \(uint8\_t address\)`](#)  
*Setting function for Tx I2C address.*
- [`void setBitRate \(uint32\_t l\_bitrate\)`](#)  
*Variable bitrate setting function.*

### Private Member Functions

- [`void initializeBus \(\)`](#)  
*I2C bus initialization.*

## Private Attributes

- `uint8_t tx_address`
- `uint32_t bitrate`

### 3.6.1 Detailed Description

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

This class manages [I2C](#) driver.

Definition at line 23 of file I2C.h.

### 3.6.2 Constructor & Destructor Documentation

#### 3.6.2.1 I2C()

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

[I2C](#) class constructor.

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

#### Parameters

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

#### Returns

Nothing

Definition at line 15 of file I2C.cpp.

Here is the call graph for this function:



### 3.6.3 Member Function Documentation

#### 3.6.3.1 initializeBus()

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

I2C bus initialization.

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

##### Returns

Nothing

Definition at line 76 of file I2C.cpp.

Here is the caller graph for this function:



#### 3.6.3.2 setBitRate()

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

Variable bitrate setting function.

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

##### Parameters

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

##### Returns

Nothing

Definition at line 71 of file I2C.cpp.

### 3.6.3.3 setTxAddress()

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

Setting function for Tx **I2C** address.

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

#### Parameters

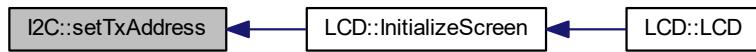
in	<i>address</i>	Requested Tx address
----	----------------	----------------------

#### Returns

Nothing

Definition at line 66 of file I2C.cpp.

Here is the caller graph for this function:



### 3.6.3.4 writeByte()

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

Byte sending function.

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

#### Parameters

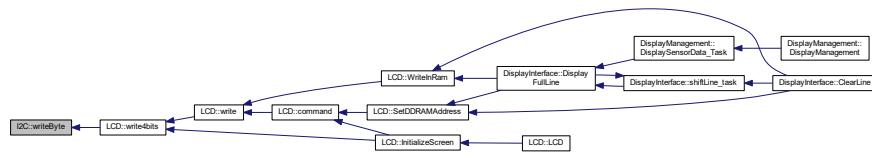
in	<i>data</i>	Pointer to the data to send
----	-------------	-----------------------------

**Returns**

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

Definition at line 23 of file I2C.cpp.

Here is the caller graph for this function:



### 3.6.4 Member Data Documentation

#### 3.6.4.1 bitrate

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

Definition at line 63 of file I2C.h.

#### 3.6.4.2 tx\_address

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

Definition at line 62 of file I2C.h.

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

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

## 3.7 keepAliveLed Class Reference

Class for keep-alive LED blinking.

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

### Public Member Functions

- [keepAliveLed \(\)](#)

*Class constructor.*

## Static Public Member Functions

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

*Task for LED blinking.*

### 3.7.1 Detailed Description

Class for keep-alive LED blinking.

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

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

### 3.7.2 Constructor & Destructor Documentation

#### 3.7.2.1 `keepAliveLed()`

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

Class constructor.

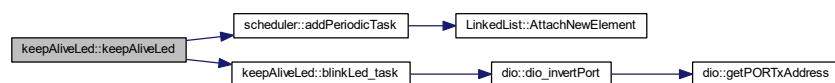
This function initializes the class `keepAliveLed`

##### Returns

Nothing

Definition at line 15 of file `keepAliveLed.cpp`.

Here is the call graph for this function:



### 3.7.3 Member Function Documentation

### 3.7.3.1 blinkLed\_task()

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

Task for LED blinking.

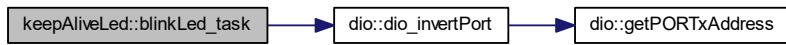
This function is inserted into the scheduler. It changes the state of the LED output to make it blink

#### Returns

Nothing

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

## 3.8 LCD Class Reference

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

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

## Public Member Functions

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

## Private Member Functions

- `void write4bits (uint8_t data)`  
*I2C write function for 4-bits mode.*
- `void write (uint8_t data, T_LCD_config_mode mode)`  
*I2C write function.*
- `void InitializeScreen ()`  
*Screen configuration function.*

## Private Attributes

- `bool backlight_enable`
- `bool cnfLineNumber`
- `bool cnfFontType`
- `bool cnfDisplayOnOff`
- `bool cnfCursorOnOff`
- `bool cnfCursorBlink`
- `bool cnfEntryModeDir`
- `bool cnfEntryModeShift`
- `uint8_t cnfI2C_addr`
- `uint8_t ddrum_addr`

### 3.8.1 Detailed Description

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

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

Definition at line 143 of file LCD.h.

### 3.8.2 Constructor & Destructor Documentation

#### 3.8.2.1 LCD()

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

[LCD](#) class constructor.

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

#### Parameters

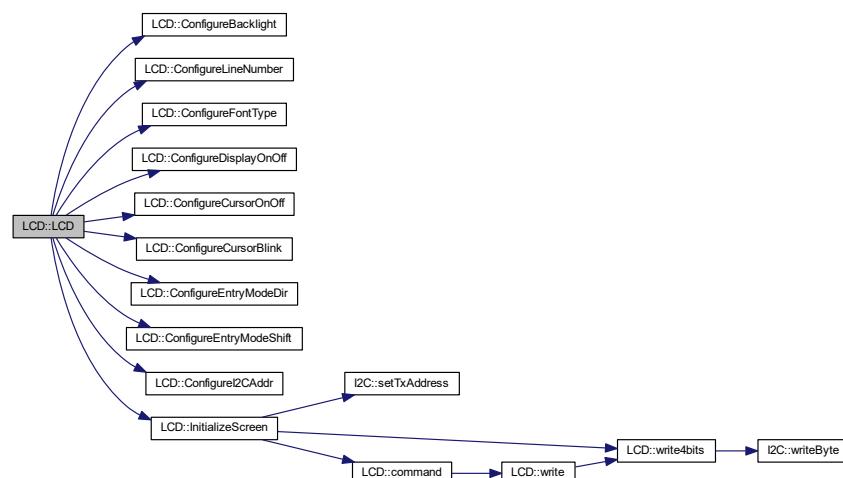
in	<i>init_conf</i>	Initial configuration structure
----	------------------	---------------------------------

#### Returns

Nothing

Definition at line 25 of file LCD.cpp.

Here is the call graph for this function:



### 3.8.3 Member Function Documentation

#### 3.8.3.1 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

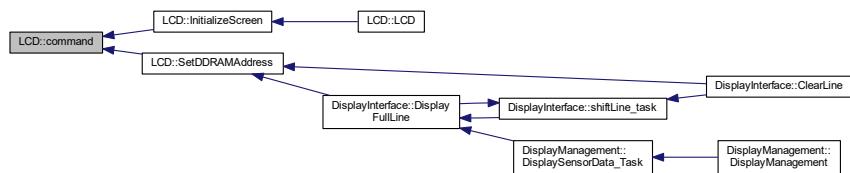
Nothing

Definition at line 126 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 3.8.3.2 ConfigureBacklight()

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

Backlight configuration function.

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

**Parameters**

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

**Returns**

Nothing

Definition at line 173 of file LCD.h.

Here is the caller graph for this function:



### 3.8.3.3 ConfigureCursorBlink()

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

Cursor blinking configuration function.

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

**Parameters**

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

**Returns**

Nothing

Definition at line 233 of file LCD.h.

Here is the caller graph for this function:



### 3.8.3.4 ConfigureCursorOnOff()

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

Cursor configuration function.

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

**Parameters**

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

**Returns**

Nothing

Definition at line 221 of file LCD.h.

Here is the caller graph for this function:



### 3.8.3.5 ConfigureDisplayOnOff()

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

Display configuration function.

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

#### Parameters

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

#### Returns

Nothing

Definition at line 209 of file LCD.h.

Here is the caller graph for this function:



### 3.8.3.6 ConfigureEntryModeDir()

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

Entry mode direction configuration function.

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

#### Parameters

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

#### Returns

Nothing

Definition at line 245 of file LCD.h.

Here is the caller graph for this function:



### 3.8.3.7 ConfigureEntryModeShift()

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

Entry mode shift configuration function.

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

#### Parameters

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

#### Returns

Nothing

Definition at line 257 of file LCD.h.

Here is the caller graph for this function:



### 3.8.3.8 ConfigureFontType()

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

Font configuration function.

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

**Parameters**

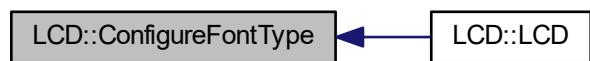
in	<i>param</i>	Configuration value
----	--------------	---------------------

**Returns**

Nothing

Definition at line 197 of file LCD.h.

Here is the caller graph for this function:



### 3.8.3.9 ConfigureI2CAddr()

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

I2C address configuration function.

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

**Parameters**

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

**Returns**

Nothing

Definition at line 269 of file LCD.h.

Here is the caller graph for this function:



### 3.8.3.10 ConfigureLineNumber()

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

Line type configuration function.

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

**Parameters**

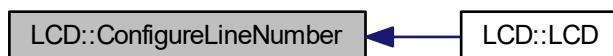
in	param	Configuration value
----	-------	---------------------

**Returns**

Nothing

Definition at line 185 of file LCD.h.

Here is the caller graph for this function:



### 3.8.3.11 GetDDRAMAddress()

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

DDRAM address get function.

This function return the value of the current DDRAM address stored in internal variable `ddram_addr`.

#### Returns

Current DDRAM address

Definition at line 289 of file LCD.h.

### 3.8.3.12 GetLineNumberCnf()

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

Number of line get function.

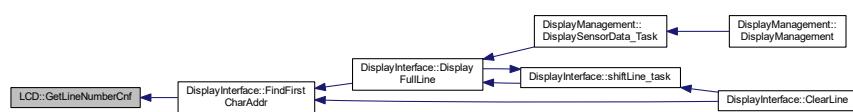
This function returns the line number configuration of the screen : 1 or 2 lines mode.

#### Returns

Line number configuration

Definition at line 311 of file LCD.h.

Here is the caller graph for this function:



## 3.8.3.13 InitializeScreen()

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

Screen configuration function.

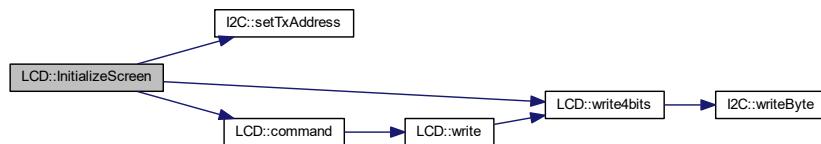
This function configures the [LCD](#) screen. It's must be called during initialization phase, or the screen won't be usable. The configuration process is described in [LCD](#) datasheet J2004A-GFDN-DYNC

#### Returns

Nothing

Definition at line 74 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 3.8.3.14 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**

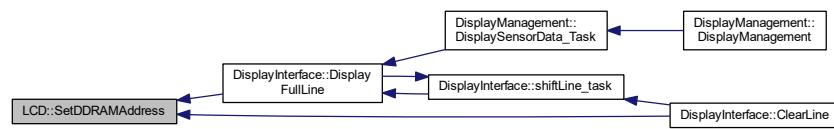
Nothing

Definition at line 169 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.8.3.15 write()**

```

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

I2C write function.

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

**Parameters**

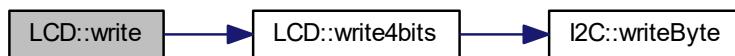
in	<i>data</i>	8-bit data for D0-7 pins of screen
in	<i>mode</i>	Requested mode for LCD communication

**Returns**

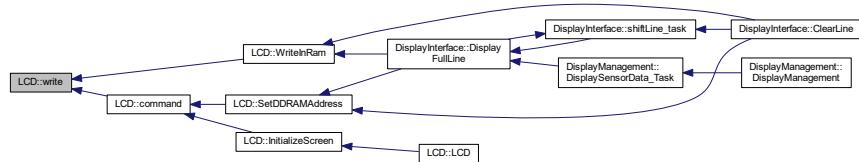
Nothing

Definition at line 63 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.8.3.16 write4bits()**

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

[I2C](#) write function for 4-bits mode.

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

**Parameters**

in	data	8-bit data to send
----	------	--------------------

**Returns**

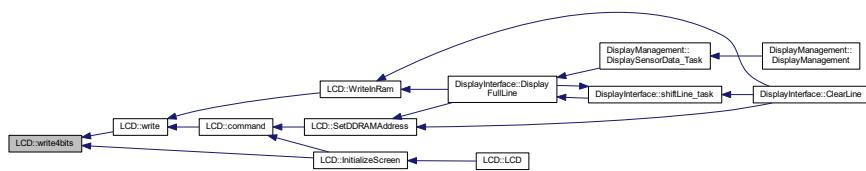
Nothing

Definition at line 46 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.8.3.17 WriteInRam()

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

Screen RAM write function.

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

#### Parameters

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

#### Returns

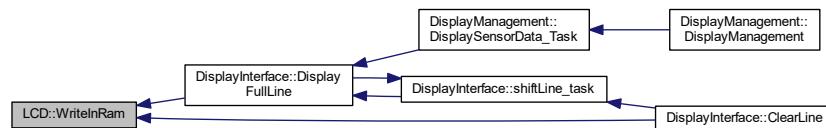
Nothing

Definition at line 191 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 3.8.4 Member Data Documentation

### 3.8.4.1 `backlight_enable`

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

Backlight enable flag

Definition at line 319 of file LCD.h.

### 3.8.4.2 `cnfCursorBlink`

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

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

Definition at line 324 of file LCD.h.

### 3.8.4.3 `cnfCursorOnOff`

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

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

Definition at line 323 of file LCD.h.

### 3.8.4.4 cnfDisplayOnOff

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

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

Definition at line 322 of file LCD.h.

### 3.8.4.5 cnfEntryModeDir

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

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

Definition at line 325 of file LCD.h.

### 3.8.4.6 cnfEntryModeShift

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

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

Definition at line 326 of file LCD.h.

### 3.8.4.7 cnfFontType

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

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

Definition at line 321 of file LCD.h.

### 3.8.4.8 cnfI2C\_addr

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

I2C address of the [LCD](#) screen

Definition at line 327 of file LCD.h.

#### 3.8.4.9 cnfLineNumber

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

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

Definition at line 320 of file LCD.h.

#### 3.8.4.10 ddram\_addr

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

Screen DDRAM address

Definition at line 329 of file LCD.h.

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

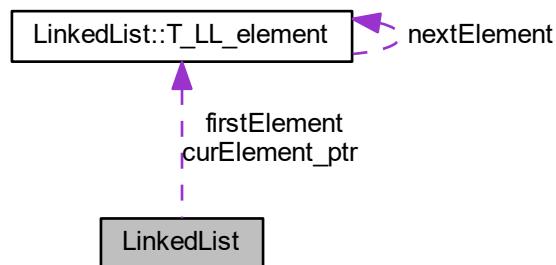
- [LCD.h](#)
- [LCD.cpp](#)

## 3.9 LinkedList Class Reference

Linked list class.

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

Collaboration diagram for LinkedList:



## Classes

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

## Public Member Functions

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

## Private Types

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

## Private Attributes

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

### 3.9.1 Detailed Description

Linked list class.

This class defines a linked list and the associated services.

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

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

### 3.9.2 Member Typedef Documentation

#### 3.9.2.1 T\_LL\_element

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

Type defining a linked list element.

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

### 3.9.3 Constructor & Destructor Documentation

#### 3.9.3.1 LinkedList()

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

Class constructor.

This constructor initializes a linked list

##### Returns

Nothing

Definition at line 12 of file LinkedList.cpp.

### 3.9.4 Member Function Documentation

#### 3.9.4.1 AttachNewElement()

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

Add an new element to the list.

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

##### Parameters

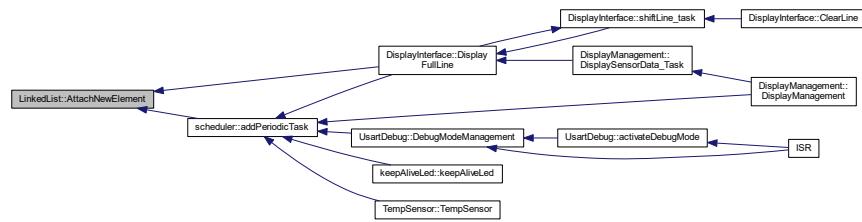
in	<i>data_ptr</i>	Pointer to the data element
----	-----------------	-----------------------------

##### Returns

Nothing

Definition at line 18 of file LinkedList.cpp.

Here is the caller graph for this function:



### 3.9.4.2 getCurrentElement()

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

Current element get function.

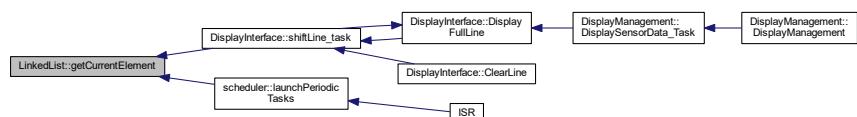
This function returns a pointer to the current pointed data in the chain.

#### Returns

Pointer to the current data

Definition at line 59 of file LinkedList.h.

Here is the caller graph for this function:



### 3.9.4.3 IsLLEmpty()

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

Empty linked list.

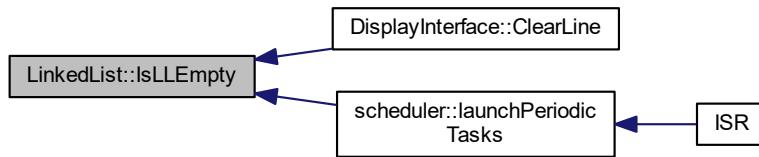
This function checks whether the linked list is empty or not (pointer to first element is equal to 0 or not).

**Returns**

True if the list is empty, false otherwise

Definition at line 84 of file LinkedList.cpp.

Here is the caller graph for this function:

**3.9.4.4 MoveToNextElement()**

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

Move to next element function.

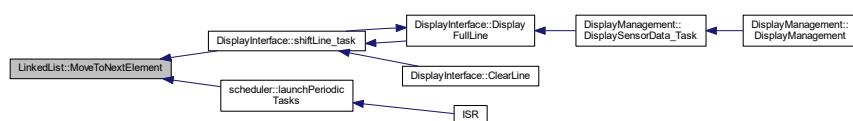
This function moves the element pointer to the next element of the chain.

**Returns**

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

Definition at line 70 of file LinkedList.cpp.

Here is the caller graph for this function:

**3.9.4.5 RemoveElement()**

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

Removes an element from the chain.

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

**Parameters**

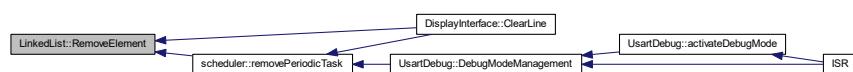
in	<i>comparisonFct_ptr</i>	Pointer to the comparison function to use
in	<i>reference_ptr</i>	Pointer to the reference data used for comparison

**Returns**

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

Definition at line 43 of file LinkedList.cpp.

Here is the caller graph for this function:

**3.9.4.6 ResetElementPtr()**

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

Resets element pointer.

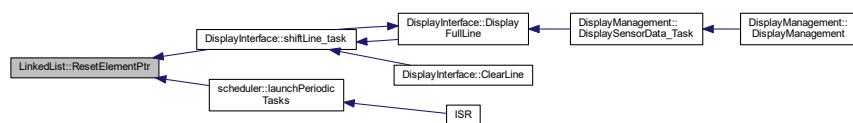
This function sets the element pointer to the first element of the chain.

**Returns**

Nothing

Definition at line 78 of file LinkedList.h.

Here is the caller graph for this function:

**3.9.5 Member Data Documentation**

### 3.9.5.1 curElement\_ptr

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

Definition at line 105 of file LinkedList.h.

### 3.9.5.2 firstElement

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

Definition at line 104 of file LinkedList.h.

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

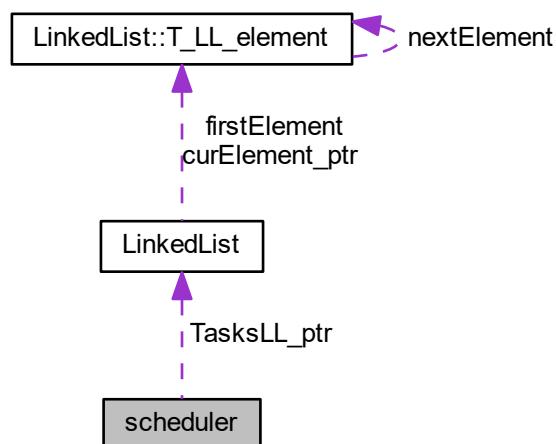
- [LinkedList.h](#)
- [LinkedList.cpp](#)

## 3.10 scheduler Class Reference

Scheduler class.

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

Collaboration diagram for scheduler:



## Classes

- struct [Task\\_t](#)

*Type defining a task structure.*

## Public Member Functions

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

## Static Public Member Functions

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

## Private Types

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

## Private Attributes

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

### 3.10.1 Detailed Description

Scheduler class.

This class defines the scheduler of the system.

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

Definition at line 30 of file scheduler.h.

### 3.10.2 Member Typedef Documentation

### 3.10.2.1 Task\_t

```
typedef struct scheduler::Task\_t scheduler::Task\_t [private]
```

Type defining a task structure.

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

## 3.10.3 Constructor & Destructor Documentation

### 3.10.3.1 scheduler()

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

scheduler class constructor

This function initializes the class scheduler

Returns

Nothing

Definition at line 30 of file scheduler.cpp.

Here is the call graph for this function:



## 3.10.4 Member Function Documentation

### 3.10.4.1 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**

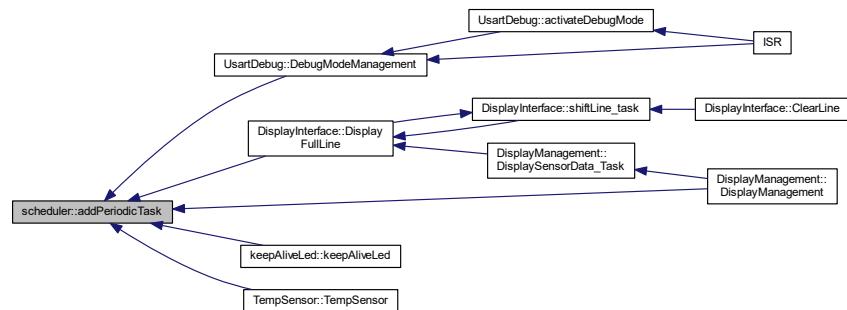
Nothing

Definition at line 83 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.10.4.2 getPitNumber()**

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

Get function for PIT number.

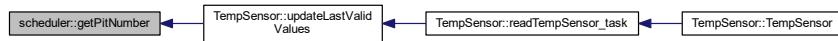
This function returns the PIT number

**Returns**

PIT number

Definition at line 95 of file scheduler.cpp.

Here is the caller graph for this function:

**3.10.4.3 launchPeriodicTasks()**

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

Main scheduler function.

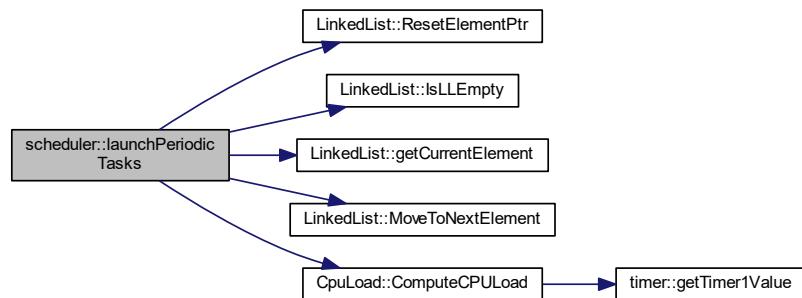
This function launches the scheduled tasks according to current software time and task configuration

**Returns**

Nothing

Definition at line 42 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.10.4.4 LLElementCompare()

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

Linked list comparison function.

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

#### Parameters

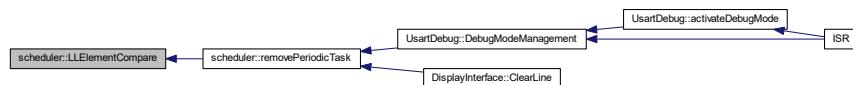
in	<i>LLElement</i>	Pointer to the linked list element
in	<i>CompareElement</i>	Pointer to the element to the compare

#### Returns

True if both elements are identical, false otherwise

Definition at line 106 of file scheduler.cpp.

Here is the caller graph for this function:



### 3.10.4.5 removePeriodicTask()

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

Remove a task from the scheduler.

This function finds the task defined by `task_ptr` in the scheduler and removes it.

#### Parameters

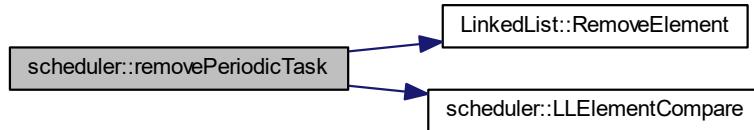
in	<i>task_ptr</i>	address of the task to remove from scheduler
----	-----------------	--

#### Returns

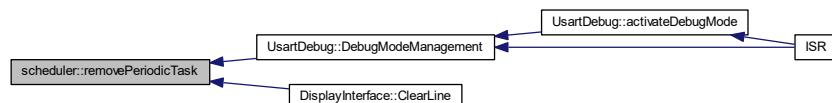
TRUE if the task has been removed, FALSE if the task does not exist in the scheduler

Definition at line 101 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 3.10.4.6 startScheduling()

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

Starts the tasks scheduling.

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

Returns

Nothing

Definition at line 77 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.10.5 Member Data Documentation

#### 3.10.5.1 pit\_number

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

Counter of periodic interrupts

Definition at line 116 of file scheduler.h.

#### 3.10.5.2 TasksLL\_ptr

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

Pointer to the linked list object containing the tasks

Definition at line 114 of file scheduler.h.

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

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

## 3.11 String Class Reference

[String](#) management class.

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

## Public Member Functions

- `String (const uint8_t *str)`  
*Class constructor.*
- `String ()`  
*Class constructor.*
- `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 Clear ()`  
*String clear function.*

## Private Member Functions

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

## Private Attributes

- `uint8_t * string`
- `uint8_t size`

### 3.11.1 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 20 of file String.h.

### 3.11.2 Constructor & Destructor Documentation

#### 3.11.2.1 `String()` [1/2]

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

Class constructor.

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

**Parameters**

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

**Returns**

Nothing

Definition at line 15 of file String.cpp.

Here is the call graph for this function:



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

`String::String ()`

Class constructor.

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

**Returns**

Nothing

Definition at line 33 of file String.cpp.

## 3.11.3 Member Function Documentation

### 3.11.3.1 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**

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

**Returns**

Nothing

Definition at line 90 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.11.3.2 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**

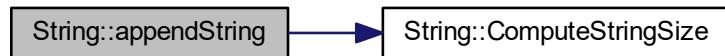
<i>in</i>	<i>str</i>	New string to add
-----------	------------	-------------------

**Returns**

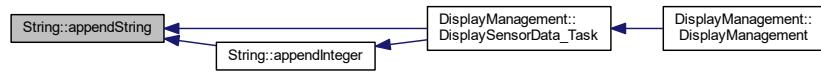
Nothing

Definition at line 52 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.11.3.3 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 108 of file String.cpp.

Here is the caller graph for this function:



### 3.11.3.4 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**

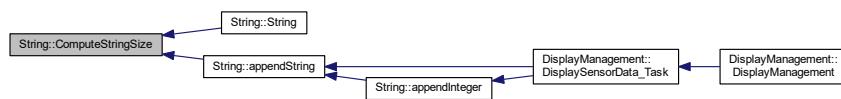
<code>in</code>	<code>str</code>	Pointer to the beginning of the string
-----------------	------------------	--

**Returns**

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

Definition at line 39 of file String.cpp.

Here is the caller graph for this function:

**3.11.3.5 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 58 of file String.h.

Here is the caller graph for this function:



### 3.11.3.6 `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 47 of file `String.h`.

Here is the caller graph for this function:



### 3.11.4 Member Data Documentation

#### 3.11.4.1 `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 95 of file `String.h`.

#### 3.11.4.2 `string`

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

Pointer to the start of the string

Definition at line 94 of file `String.h`.

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

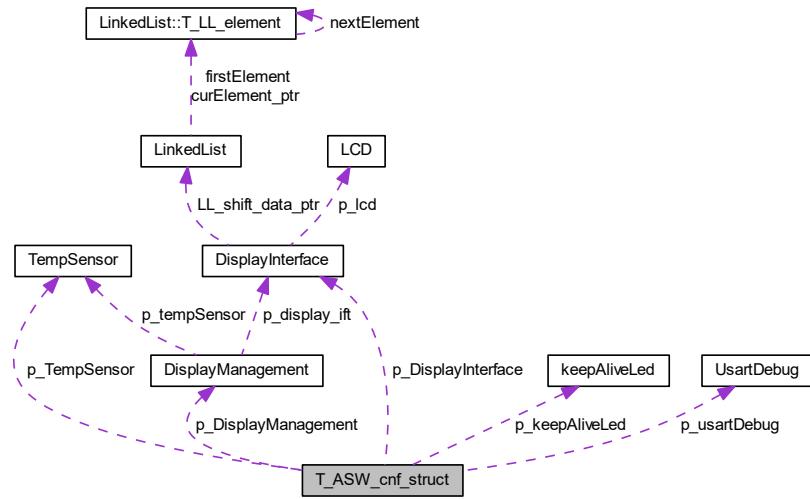
- [String.h](#)
- [String.cpp](#)

## 3.12 T\_ASW\_cnf\_struct Struct Reference

ASW configuration structure.

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

Collaboration diagram for T\_ASW\_cnf\_struct:



### Public Attributes

- `UsartDebug * p_usartDebug`
- `keepAliveLed * p_keepAliveLed`
- `TempSensor * p_TempSensor`
- `DisplayInterface * p_DisplayInterface`
- `DisplayManagement * p_DisplayManagement`

#### 3.12.1 Detailed Description

ASW configuration structure.

This structure contains all pointers to instanced applicative objects

Definition at line 18 of file asw.h.

#### 3.12.2 Member Data Documentation

### 3.12.2.1 p\_DisplayInterface

`DisplayInterface* T_ASW_cnf_struct::p_DisplayInterface`

Pointer to [DisplayInterface](#) object

Definition at line 23 of file asw.h.

### 3.12.2.2 p\_DisplayManagement

`DisplayManagement* T_ASW_cnf_struct::p_DisplayManagement`

Pointer to [DisplayManagement](#) object

Definition at line 24 of file asw.h.

### 3.12.2.3 p\_keepAliveLed

`keepAliveLed* T_ASW_cnf_struct::p_keepAliveLed`

Pointer to [keepAliveLed](#) object

Definition at line 21 of file asw.h.

### 3.12.2.4 p\_TempSensor

`TempSensor* T_ASW_cnf_struct::p_TempSensor`

Pointer to [TempSensor](#) object

Definition at line 22 of file asw.h.

### 3.12.2.5 p\_usartDebug

`UsartDebug* T_ASW_cnf_struct::p_usartDebug`

Pointer to USART debug object

Definition at line 20 of file asw.h.

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

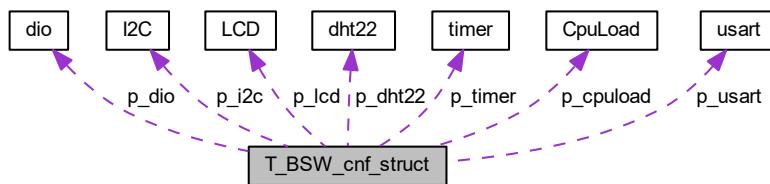
- [asw.h](#)

### 3.13 T\_BSW\_cnf\_struct Struct Reference

BSW configuration structure.

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

Collaboration diagram for T\_BSW\_cnf\_struct:



#### Public Attributes

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

#### 3.13.1 Detailed Description

BSW configuration structure.

This structure contains all pointers to instanced drivers objects

Definition at line 24 of file bsw.h.

#### 3.13.2 Member Data Documentation

##### 3.13.2.1 p\_cpuload

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

Pointer to cpu load library object

Definition at line 30 of file bsw.h.

### 3.13.2.2 p\_dht22

`dht22* T_BSW_cnf_struct::p_dht22`

Pointer to [dht22](#) driver object

Definition at line 29 of file bsw.h.

### 3.13.2.3 p\_dio

`dio* T_BSW_cnf_struct::p_dio`

Pointer to dio driver object

Definition at line 27 of file bsw.h.

### 3.13.2.4 p\_i2c

`I2C* T_BSW_cnf_struct::p_i2c`

Pointer to [I2C](#) driver object

Definition at line 31 of file bsw.h.

### 3.13.2.5 p\_lcd

`LCD* T_BSW_cnf_struct::p_lcd`

Pointer to [LCD](#) driver object

Definition at line 32 of file bsw.h.

### 3.13.2.6 p\_timer

`timer* T_BSW_cnf_struct::p_timer`

Pointer to timer driver object

Definition at line 28 of file bsw.h.

### 3.13.2.7 p\_usart

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

Pointer to usart driver object

Definition at line 26 of file bsw.h.

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

- [bsw.h](#)

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

Structure containing shift data.

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

### Public Attributes

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

### 3.14.1 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 33 of file DisplayInterface.h.

### 3.14.2 Member Data Documentation

#### 3.14.2.1 line

```
uint8_t T_Display_shift_data::line
```

Definition at line 38 of file DisplayInterface.h.

### 3.14.2.2 size

```
uint8_t T_Display_shift_data::size
```

Definition at line 37 of file DisplayInterface.h.

### 3.14.2.3 str\_cur\_ptr

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

Definition at line 36 of file DisplayInterface.h.

### 3.14.2.4 str\_start\_ptr

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

Definition at line 35 of file DisplayInterface.h.

### 3.14.2.5 temporization

```
uint8_t T_Display_shift_data::temporization
```

Definition at line 39 of file DisplayInterface.h.

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

- [DisplayInterface.h](#)

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

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

### Public Attributes

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

### 3.15.1 Detailed Description

Definition at line 125 of file LCD.h.

### 3.15.2 Member Data Documentation

#### 3.15.2.1 `backlight_en`

```
bool T_LCD_conf_struct::backlight_en
```

Definition at line 128 of file LCD.h.

#### 3.15.2.2 `cursor_en`

```
bool T_LCD_conf_struct::cursor_en
```

Definition at line 132 of file LCD.h.

#### 3.15.2.3 `cursorBlink_en`

```
bool T_LCD_conf_struct::cursorBlink_en
```

Definition at line 133 of file LCD.h.

#### 3.15.2.4 `display_en`

```
bool T_LCD_conf_struct::display_en
```

Definition at line 131 of file LCD.h.

#### 3.15.2.5 `entryModeDir`

```
bool T_LCD_conf_struct::entryModeDir
```

Definition at line 134 of file LCD.h.

### 3.15.2.6 entryModeShift

```
bool T_LCD_conf_struct::entryModeShift
```

Definition at line 135 of file LCD.h.

### 3.15.2.7 fontType\_cnf

```
bool T_LCD_conf_struct::fontType_cnf
```

Definition at line 130 of file LCD.h.

### 3.15.2.8 i2c\_addr

```
uint8_t T_LCD_conf_struct::i2c_addr
```

Definition at line 127 of file LCD.h.

### 3.15.2.9 lineNumber\_cnf

```
bool T_LCD_conf_struct::lineNumber_cnf
```

Definition at line 129 of file LCD.h.

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

- [LCD.h](#)

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

Type defining a linked list element.

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



## Public Attributes

- `void * data_ptr`
- `T_LL_element * nextElement`

### 3.16.1 Detailed Description

Type defining a linked list element.

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

Definition at line 97 of file `LinkedList.h`.

### 3.16.2 Member Data Documentation

#### 3.16.2.1 `data_ptr`

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

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

#### 3.16.2.2 `nextElement`

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

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

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

- [LinkedList.h](#)

## 3.17 `scheduler::Task_t` Struct Reference

Type defining a task structure.

## Public Attributes

- `TaskPtr_t TaskPtr`
- `uint16_t period`

### 3.17.1 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 107 of file scheduler.h.

### 3.17.2 Member Data Documentation

#### 3.17.2.1 period

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

Period of the task

Definition at line 110 of file scheduler.h.

#### 3.17.2.2 TaskPtr

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

Pointer to the task

Definition at line 109 of file scheduler.h.

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

- [scheduler.h](#)

## 3.18 TempSensor Class Reference

Class for temperature sensor.

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

## Public Member Functions

- `TempSensor ()`  
*Class constructor.*
- `uint16_t * getTempPtr ()`  
*Get pointer to data raw\_temperature.*
- `uint16_t * getHumPtr ()`  
*Get pointer to data raw\_humidity.*
- `bool getTemp (uint16_t *temp)`  
*Get temperature data.*
- `bool getHumidity (uint16_t *hum)`  
*Get humidity data.*
- `void setValidity (bool validity)`  
*Set data val\_validity.*
- `void updateLastValidValues ()`
- `uint8_t GetTempInteger ()`  
*Temperature formatting function - Integer part.*
- `uint8_t GetTempDecimal ()`  
*Temperature formatting function - Decimal part.*
- `uint8_t GetHumInteger ()`  
*Humidity formatting function - Integer part.*
- `uint8_t GetHumDecimal ()`  
*Humidity formatting function - Decimal part.*
- `bool GetValidity ()`  
*Data validity get function.*

## Static Public Member Functions

- `static void readTempSensor_task ()`  
*Task for reading temperature and humidity values.*

## Private Attributes

- `uint16_t read_temperature`
- `uint16_t read_humidity`
- `bool validity_last_read`
- `bool validity`
- `uint32_t valid_pit`
- `uint16_t valid_temp`
- `uint16_t valid_hum`

### 3.18.1 Detailed Description

Class for temperature sensor.

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

Definition at line 19 of file TempSensor.h.

### 3.18.2 Constructor & Destructor Documentation

### 3.18.2.1 TempSensor()

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

## Class constructor.

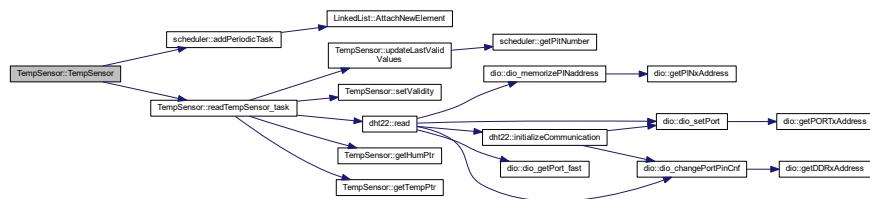
This function initializes all data of the class [TempSensor](#)

## Returns

Nothing

Definition at line 16 of file TempSensor.cpp.

Here is the call graph for this function:



### 3.18.3 Member Function Documentation

### 3.18.3.1 GetHumDecimal()

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

Humidity formatting function - Decimal part.

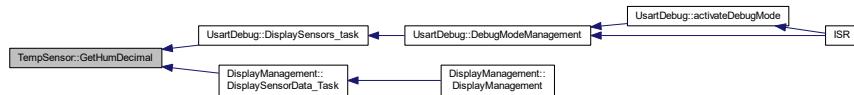
This function return the decimal part of the humidity

## Returns

Decimal value of the humidity

Definition at line 122 of file TempSensor.h.

Here is the caller graph for this function:



### 3.18.3.2 getHumidity()

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

Get humidity data.

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

#### Parameters

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

#### Returns

Validity of humidity

Definition at line 66 of file TempSensor.cpp.

### 3.18.3.3 GetHumInteger()

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

Humidity formatting function - Integer part.

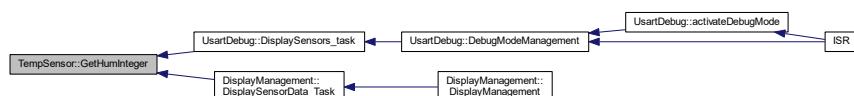
This function return the integer part of the humidity

#### Returns

Integer value of the humidity

Definition at line 111 of file TempSensor.h.

Here is the caller graph for this function:



### 3.18.3.4 getHumPtr()

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

Get pointer to data raw\_humidity.

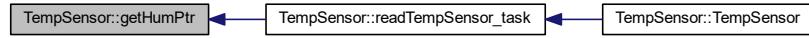
This function returns a pointer to the class member raw\_humidity

#### Returns

Pointer to raw\_humidity

Definition at line 41 of file TempSensor.cpp.

Here is the caller graph for this function:



### 3.18.3.5 getTemp()

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

Get temperature data.

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

#### Parameters

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

#### Returns

Validity of temperature

Definition at line 72 of file TempSensor.cpp.

### 3.18.3.6 GetTempDecimal()

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

Temperature formatting function - Decimal part.

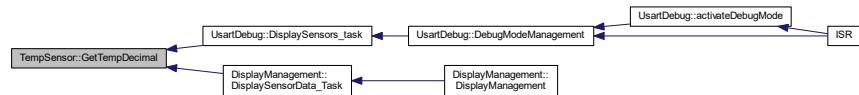
This function return the decimal part of the temperature

**Returns**

Decimal value of the temperature

Definition at line 100 of file TempSensor.h.

Here is the caller graph for this function:

**3.18.3.7 GetTempInteger()**

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

Temperature formatting function - Integer part.

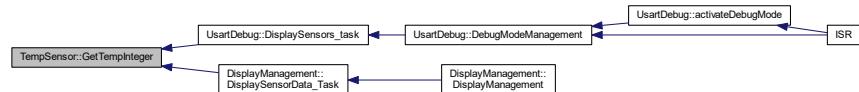
This function return the integer part of the temperature

**Returns**

Integer value of the temperature

Definition at line 89 of file TempSensor.h.

Here is the caller graph for this function:



### 3.18.3.8 getTempPtr()

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

Get pointer to data raw\_temperature.

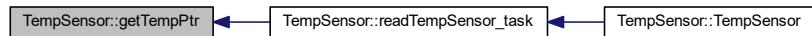
This function returns a pointer to the class member raw\_temperature

#### Returns

Pointer to raw\_temperature

Definition at line 46 of file TempSensor.cpp.

Here is the caller graph for this function:



### 3.18.3.9 GetValidity()

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

Data validity get function.

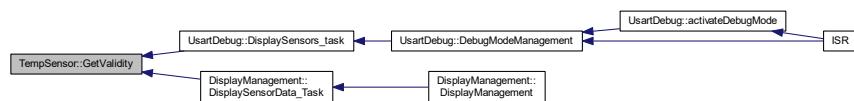
This function returns the validity of the sensor data

#### Returns

True if the sensor values are valid, false otherwise

Definition at line 133 of file TempSensor.h.

Here is the caller graph for this function:



### 3.18.3.10 `readTempSensor_task()`

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

Task for reading temperature and humidity values.

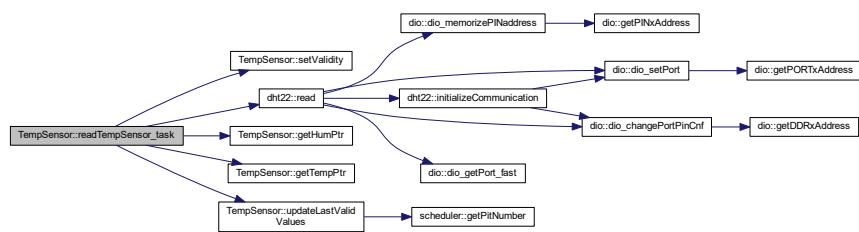
This task reads temperature and humidity data using DHT22 driver. It is called every 5 seconds.

#### Returns

Nothing

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

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.18.3.11 `setValidity()`

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

Set data `val_validity`.

This function sets the class member `val_validity`

#### Parameters

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

Returns

Nothing

Definition at line 36 of file TempSensor.cpp.

Here is the caller graph for this function:

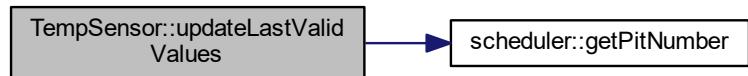


### 3.18.3.12 updateLastValidValues()

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

Definition at line 51 of file TempSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 3.18.4 Member Data Documentation

### 3.18.4.1 `read_humidity`

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

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

Definition at line 141 of file TempSensor.h.

### 3.18.4.2 `read_temperature`

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

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

Definition at line 140 of file TempSensor.h.

### 3.18.4.3 `valid_hum`

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

Valid value of humidity

Definition at line 148 of file TempSensor.h.

### 3.18.4.4 `valid_pit`

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

pit number of the last time when data were valid

Definition at line 145 of file TempSensor.h.

### 3.18.4.5 `valid_temp`

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

Valid value of temperature

Definition at line 147 of file TempSensor.h.

#### 3.18.4.6 validity

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

validity of official temperature and humidity data

Definition at line 144 of file TempSensor.h.

#### 3.18.4.7 validity\_last\_read

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

Validity of last read temperature and humidity data

Definition at line 142 of file TempSensor.h.

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

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

## 3.19 timer Class Reference

Class defining a timer.

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

### Public Member Functions

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

### Private Attributes

- uint8\_t [prescaler](#)

### 3.19.1 Detailed Description

Class defining a timer.

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

Definition at line 19 of file timer.h.

### 3.19.2 Constructor & Destructor Documentation

#### 3.19.2.1 timer()

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

Class constructor.

This function initializes class attributes

Returns

Nothing

Definition at line 13 of file timer.cpp.

### 3.19.3 Member Function Documentation

#### 3.19.3.1 configureTimer1()

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

Configures Timer #1.

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

Parameters

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

**Returns**

Nothing

Definition at line 18 of file timer.cpp.

Here is the caller graph for this function:



### 3.19.3.2 getTimer1Value()

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

Reads current value of timer #1.

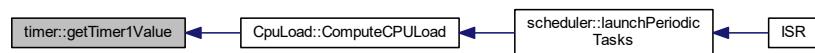
This function reads the value of of timer #1 using register TCNT1. The function is inlined to speed up SW execution.

**Returns**

Current timer value

Definition at line 58 of file timer.h.

Here is the caller graph for this function:



### 3.19.3.3 startTimer1()

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

Start Timer #1.

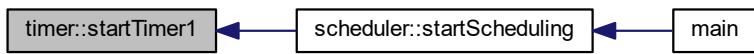
This functions starts Timer #1. Timer shall be initialized before this function is called.

#### Returns

Nothing

Definition at line 56 of file timer.cpp.

Here is the caller graph for this function:



### 3.19.3.4 stopTimer1()

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

Stops Timer #1.

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

#### Returns

Nothing

Definition at line 67 of file timer.cpp.

## 3.19.4 Member Data Documentation

### 3.19.4.1 prescaler

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

Definition at line 64 of file timer.h.

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

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

## 3.20 usart Class Reference

USART serial bus class.

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

### Public Member Functions

- [uart](#) (uint16\_t a\_BaudRate)  
*Class usart constructor.*
- void [uart\\_sendString](#) (uint8\_t \*str)  
*Sending a string on USART link.*
- void [setBaudRate](#) (uint16\_t a\_BaudRate)  
*Setting baud rate.*
- void [uart\\_init](#) ()  
*USART hardware initialization.*
- uint8\_t [uart\\_read](#) ()  
*USART read function.*

### Private Member Functions

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

### Private Attributes

- uint16\_t [BaudRate](#)

#### 3.20.1 Detailed Description

USART serial bus class.

This class defines all useful functions for USART serial bus

Definition at line 16 of file usart.h.

#### 3.20.2 Constructor & Destructor Documentation

##### 3.20.2.1 usart()

```
uart::uart (
    uint16_t a_BaudRate )
```

Class usart constructor.

Initializes the class and call hardware initialization function

**Parameters**

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

**Returns**

Nothing.

Definition at line 14 of file usart.cpp.

Here is the call graph for this function:



### 3.20.3 Member Function Documentation

#### 3.20.3.1 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 63 of file usart.cpp.

### 3.20.3.2 usart\_init()

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

USART hardware initialization.

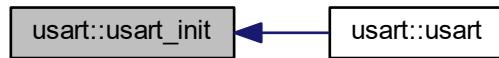
This function will initialize the USART using selected baudrate. User must pay attention to select one of the usually used Baud Rate (9600, 19200, 38400, 57600). Note that since an uint16 is used as argument, Baud rate cannot be more than 57600.

#### Returns

Nothing.

Definition at line 21 of file usart.cpp.

Here is the caller graph for this function:



### 3.20.3.3 usart\_read()

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

USART read function.

This function will read reception register of USART

#### Returns

The function returns the 8 bits read from reception buffer

Definition at line 79 of file usart.cpp.

Here is the caller graph for this function:



### 3.20.3.4 usart\_sendString()

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

Sending a string on USART link.

Just write data to the Serial link using usart\_transmit function

#### Parameters

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

#### Returns

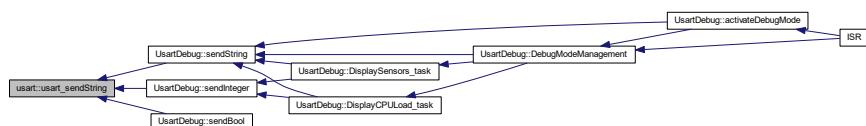
Nothing.

Definition at line 44 of file usart.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.20.3.5 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**

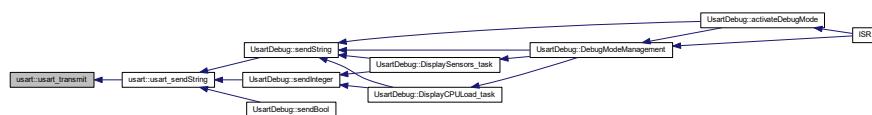
in	<i>Data</i>	Desired data char to transmit
----	-------------	-------------------------------

**Returns**

Nothing.

Definition at line 70 of file usart.cpp.

Here is the caller graph for this function:



### 3.20.4 Member Data Documentation

#### 3.20.4.1 BaudRate

```
uint16_t USART::BaudRate [private]
```

Defines the baud rate used by driver

Definition at line 70 of file usart.h.

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

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

## 3.21 UsartDebug Class Reference

Class used for debugging on usart link.

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

## Public Member Functions

- [UsartDebug \(\)](#)  
*Class UsartDebug constructor.*
- [void sendInteger \(uint16\\_t data, uint8\\_t base\)](#)  
*Send a integer data on USART link.*
- [void sendBool \(bool data\)](#)  
*Send a boolean data on USART link.*
- [void sendString \(uint8\\_t \\*str\)](#)  
*Send a string on USART link.*
- [bool isDebugModeActive \(\)](#)  
*Check is debug mode is active or not.*
- [void activateDebugMode \(\)](#)  
*Activates debug mode.*
- [void DebugModeManagement \(uint8\\_t rcv\\_char\)](#)  
*Management of debug mode.*

## Static Public Member Functions

- [static void DisplaySensors\\_task \(\)](#)  
*Displays sensors data on usart link.*
- [static void DisplayCPUload\\_task \(\)](#)  
*Displays CPU load data on usart link.*

## Private Attributes

- [debug\\_state\\_t debug\\_state](#)
- [bool debugModeActive\\_F](#)

### 3.21.1 Detailed Description

Class used for debugging on usart link.

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

Definition at line 34 of file debug.h.

### 3.21.2 Constructor & Destructor Documentation

#### 3.21.2.1 UsartDebug()

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

Class [UsartDebug](#) constructor.

Initializes the class [UsartDebug](#)

#### Returns

Nothing

Definition at line 54 of file debug.cpp.

### 3.21.3 Member Function Documentation

#### 3.21.3.1 activateDebugMode()

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

Activates debug mode.

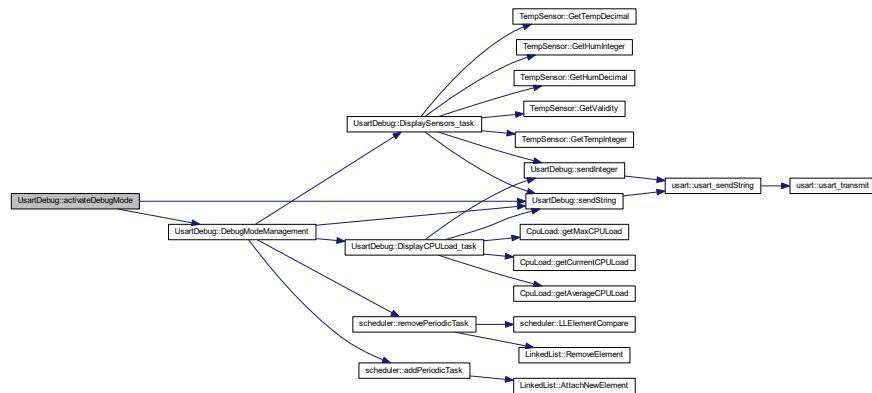
This function activates USART debug mode.

#### Returns

Nothing

Definition at line 142 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.21.3.2 DebugModeManagement()

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

Management of debug mode.

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

- init state : display main menu
- WAIT\_INIT state : handles user choice in main menu and selects next state
- DISPLAY\_DATA state : display sensor data periodically
- DISPLAY\_CPU\_LOAD : display CPU load periodically

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

#### Parameters

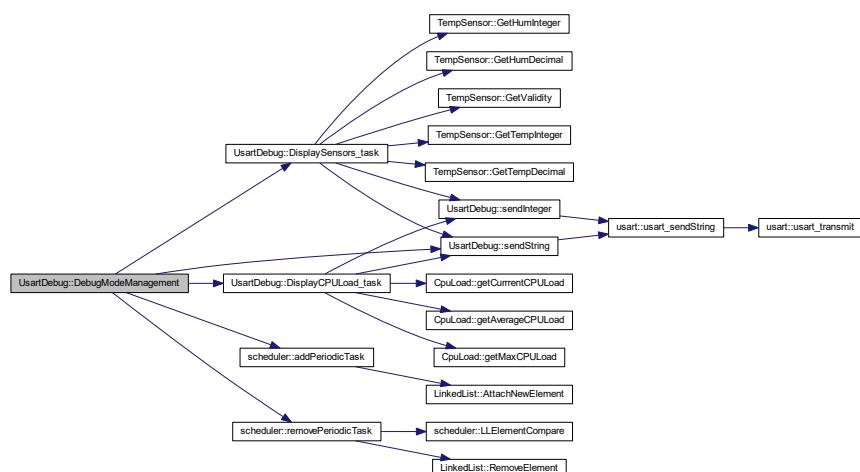
in	<i>rcv_char</i>	8 bits character received on USART
----	-----------------	------------------------------------

#### Returns

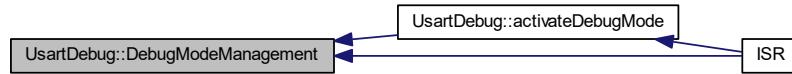
Nothing

Definition at line 150 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.21.3.3 DisplayCPUload\_task()

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

Displays CPU load data on usart link.

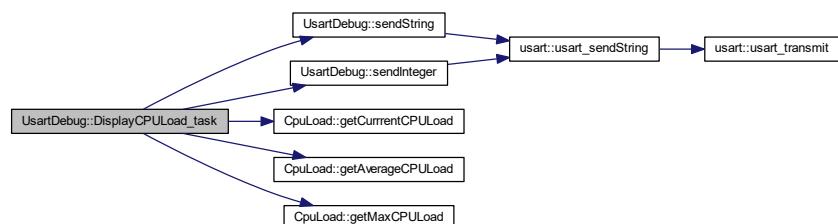
This task sends CPU load data (current and average load) on usart link every 5 seconds

#### Returns

Nothing

Definition at line 126 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.21.3.4 DisplaySensors\_task()

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

Displays sensors data on uart link.

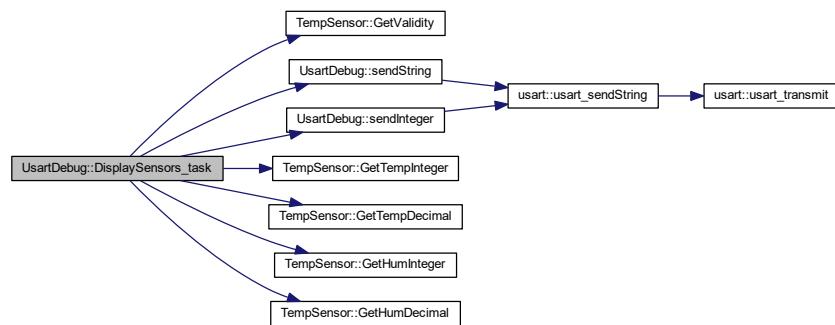
This task sends sensors data (temperature and humidity) on uart link every 5 seconds

#### Returns

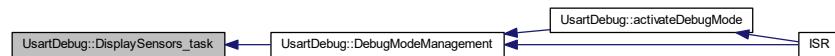
Nothing

Definition at line 94 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.21.3.5 isDebugEnabled()

```
bool UsartDebug::isDebugEnabled ( )
```

Check if debug mode is active or not.

This function checks if debug mode is active or not.

**Returns**

TRUE if debug mode is active, FALSE otherwise

Definition at line 137 of file debug.cpp.

Here is the caller graph for this function:



### 3.21.3.6 sendBool()

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

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.

**Parameters**

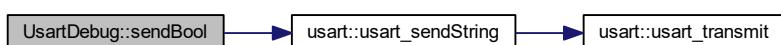
in	<i>data</i>	boolean data to be sent
----	-------------	-------------------------

**Returns**

Nothing

Definition at line 80 of file debug.cpp.

Here is the call graph for this function:



### 3.21.3.7 sendInteger()

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

Send a integer data on USART link.

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

#### Parameters

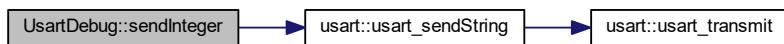
in	<i>data</i>	integer data to be sent
in	<i>base</i>	numerical base used to convert integer into string (between 2 and 36)

#### Returns

Nothing

Definition at line 66 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.21.3.8 sendString()

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

Send a string on USART link.

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

**Parameters**

in	str	Pointer to the string being sent
----	-----	----------------------------------

**Returns**

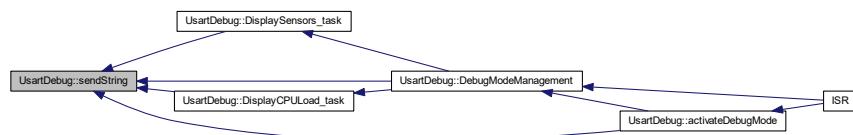
Nothing

Definition at line 60 of file debug.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.21.4 Member Data Documentation

#### 3.21.4.1 debug\_state

`debug_state_t UsartDebug::debug_state [private]`

Definition at line 117 of file debug.h.

#### 3.21.4.2 debugModeActive\_F

`bool UsartDebug::debugModeActive_F [private]`

Debug mode activation flag

Definition at line 118 of file debug.h.

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

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



## Chapter 4

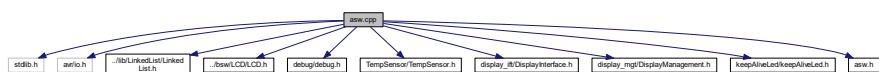
# File Documentation

### 4.1 asw.cpp File Reference

ASW main file.

```
#include <stdlib.h>
#include <avr/io.h>
#include "../lib/LinkedList/LinkedList.h"
#include "../bsw/LCD/LCD.h"
#include "debug/debug.h"
#include "TempSensor/TempSensor.h"
#include "display_ift/DisplayInterface.h"
#include "display_mgt/DisplayManagement.h"
#include "keepAliveLed/keepAliveLed.h"
#include "asw.h"
```

Include dependency graph for asw.cpp:



### Functions

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

*Initialization of ASW.*

### Variables

- [T\\_ASW\\_cnf\\_struct ASW\\_cnf\\_struct](#)

#### 4.1.1 Detailed Description

ASW main file.

##### Date

15 mars 2018

##### Author

nicls67

#### 4.1.2 Function Documentation

##### 4.1.2.1 asw\_init()

```
void asw_init ( )
```

Initialization of ASW.

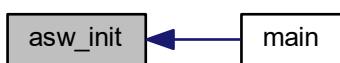
This function instantiates all applicative objects. The addresses of objects are then stored in ASW\_cnf\_struct structure. This function shall be called after BSW initialization function.

##### Returns

Nothing

Definition at line 30 of file asw.cpp.

Here is the caller graph for this function:



#### 4.1.3 Variable Documentation

### 4.1.3.1 ASW\_cnf\_struct

`T_ASW_cnf_struct ASW_cnf_struct`

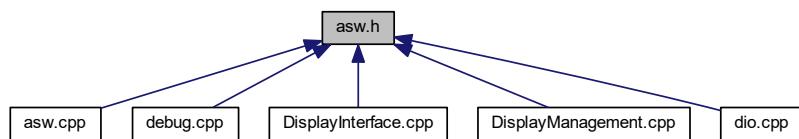
ASW configuration structure

Definition at line 27 of file asw.cpp.

## 4.2 asw.h File Reference

ASW main header file.

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



### Classes

- struct `T_ASW_cnf_struct`  
*ASW configuration structure.*

### Functions

- void `asw_init()`  
*Initialization of ASW.*

### Variables

- `T_ASW_cnf_struct ASW_cnf_struct`

### 4.2.1 Detailed Description

ASW main header file.

#### Date

15 mars 2018

#### Author

nicls67

## 4.2.2 Function Documentation

### 4.2.2.1 asw\_init()

```
void asw_init ( )
```

Initialization of ASW.

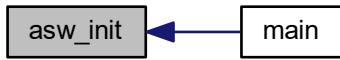
This function instantiates all applicative objects. The addresses of objects are then stored in ASW\_cnf\_struct structure. This function shall be called after BSW initialization function.

#### Returns

Nothing

Definition at line 30 of file asw.cpp.

Here is the caller graph for this function:



## 4.2.3 Variable Documentation

### 4.2.3.1 ASW\_cnf\_struct

```
T_ASW_cnf_struct ASW_cnf_struct
```

ASW configuration structure

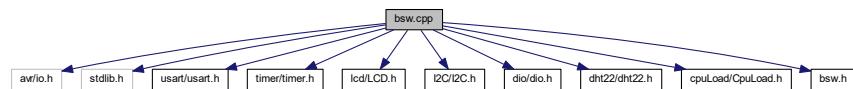
Definition at line 27 of file asw.cpp.

## 4.3 bsw.cpp File Reference

BSW main file.

```
#include <avr/io.h>
#include <stdlib.h>
#include "uart/usart.h"
#include "timer/timer.h"
#include "lcd/LCD.h"
#include "I2C/I2C.h"
#include "dio/dio.h"
#include "dht22/dht22.h"
#include "cpuLoad/CpuLoad.h"
#include "bsw.h"
```

Include dependency graph for bsw.cpp:



## Functions

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

*Initialization of BSW.*

## Variables

- [T\\_BSW\\_cnf\\_struct BSW\\_cnf\\_struct](#)

### 4.3.1 Detailed Description

BSW main file.

#### Date

13 mars 2018

#### Author

nicls67

### 4.3.2 Function Documentation

#### 4.3.2.1 bsw\_init()

```
void bsw_init ( )
```

Initialization of BSW.

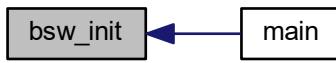
This function instantiates all driver objects, leading hardware initialization. The addresses of driver objects are then stored in BSW\_cnf\_struct structure.

Returns

Nothing

Definition at line 25 of file bsw.cpp.

Here is the caller graph for this function:



#### 4.3.3 Variable Documentation

##### 4.3.3.1 BSW\_cnf\_struct

`T_BSW_cnf_struct BSW_cnf_struct`

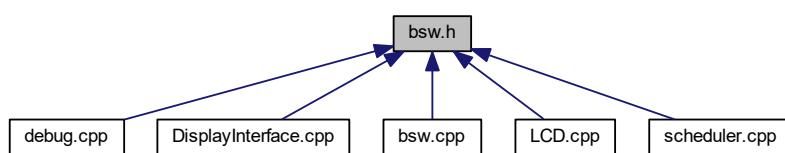
BSW configuration structure

Definition at line 23 of file bsw.cpp.

#### 4.4 bsw.h File Reference

BSW main header file.

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



## Classes

- struct [T\\_BSW\\_cnf\\_struct](#)

*BSW configuration structure.*

## Macros

- #define [USART\\_BAUDRATE](#) (uint16\_t)9600
- #define [I2C\\_BITRATE](#) (uint32\_t)100000

## Functions

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

*Initialization of BSW.*

## Variables

- [T\\_BSW\\_cnf\\_struct BSW\\_cnf\\_struct](#)

### 4.4.1 Detailed Description

BSW main header file.

#### Date

13 mars 2018

#### Author

nicls67

### 4.4.2 Macro Definition Documentation

#### 4.4.2.1 I2C\_BITRATE

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

[I2C bus bitrate is 100 kHz](#)

Definition at line 18 of file bsw.h.

#### 4.4.2.2 USART\_BAUDRATE

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

uart connection to PC uses a baud rate of 9600

Definition at line 17 of file bsw.h.

### 4.4.3 Function Documentation

#### 4.4.3.1 bsw\_init()

```
void bsw_init ( )
```

Initialization of BSW.

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

#### Returns

Nothing

Definition at line 25 of file bsw.cpp.

Here is the caller graph for this function:



### 4.4.4 Variable Documentation

#### 4.4.4.1 BSW\_cnf\_struct

```
T_BSW_cnf_struct BSW_cnf_struct
```

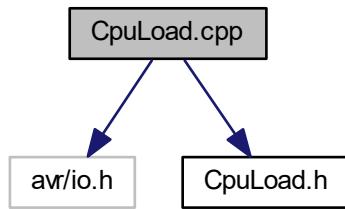
BSW configuration structure

Definition at line 23 of file bsw.cpp.

## 4.5 CpuLoad.cpp File Reference

Defines functions of class [CpuLoad](#).

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



### 4.5.1 Detailed Description

Defines functions of class [CpuLoad](#).

#### Date

21 mars 2019

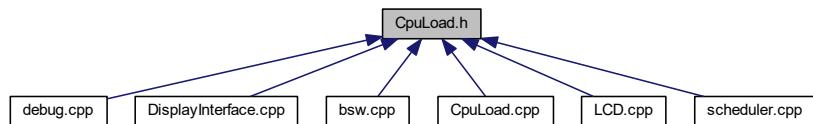
#### Author

nicls67

## 4.6 CpuLoad.h File Reference

[CpuLoad](#) class header file.

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



## Classes

- class [CpuLoad](#)

*Class defining CPU load libraries.*

## Macros

- `#define NB_OF_SAMPLES 50`

### 4.6.1 Detailed Description

[CpuLoad](#) class header file.

#### Date

21 mars 2019

#### Author

nicls67

### 4.6.2 Macro Definition Documentation

#### 4.6.2.1 NB\_OF\_SAMPLES

```
#define NB_OF_SAMPLES 50
```

Definition at line 13 of file CpuLoad.h.

## 4.7 debug.cpp File Reference

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

```
#include <avr/io.h>
#include <stdlib.h>
#include "../../lib/LinkedList/LinkedList.h"
#include "../../scheduler/scheduler.h"
#include "../../bsw/usart/usart.h"
#include "../../bsw/timer/timer.h"
#include "../../bsw/lcd/LCD.h"
#include "../../bsw/I2C/I2C.h"
#include "../../bsw/dio/dio.h"
#include "../../bsw/dht22/dht22.h"
#include "../../bsw/cpuLoad/CpuLoad.h"
#include "../../bsw/bsw.h"
#include "../debug/debug.h"
#include "../TempSensor/TempSensor.h"
#include "../display_ift/DisplayInterface.h"
#include "../display_mgt/DisplayManagement.h"
#include "../keepAliveLed/keepAliveLed.h"
#include "../asw.h"
```

Include dependency graph for debug.cpp:



## Variables

- const char str\_debug\_main\_menu []

*Main menu of debug mode.*

### 4.7.1 Detailed Description

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

#### Date

15 mars 2018

#### Author

nicls67

### 4.7.2 Variable Documentation

#### 4.7.2.1 str\_debug\_main\_menu

```
const char str_debug_main_menu[]
```

##### Initial value:

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

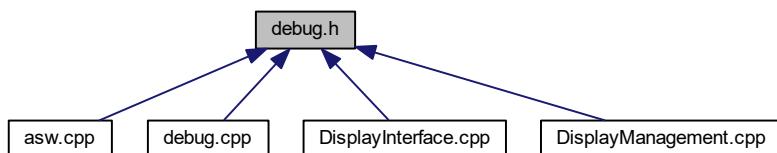
Main menu of debug mode.

Definition at line 43 of file debug.cpp.

## 4.8 debug.h File Reference

Header file for debug and logging functions.

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



## Classes

- class [UsartDebug](#)

*Class used for debugging on usart link.*

## Macros

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

## Enumerations

- enum [debug\\_state\\_t](#) { `INIT`, `WAIT_INIT`, `DISPLAY_DATA`, `DISPLAY_CPU_LOAD` }

*Defines the debug states.*

### 4.8.1 Detailed Description

Header file for debug and logging functions.

#### Date

15 mars 2018

#### Author

nicls67

### 4.8.2 Macro Definition Documentation

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

```
#define PERIOD_MS_TASK_DISPLAY_CPU_LOAD 5000
```

Period for displaying CPU load data

Definition at line 14 of file debug.h.

#### 4.8.2.2 PERIOD\_MS\_TASK\_DISPLAY\_SENSORS

```
#define PERIOD_MS_TASK_DISPLAY_SENSORS 5000
```

Period for displaying temperature and humidity data

Definition at line 13 of file debug.h.

### 4.8.3 Enumeration Type Documentation

#### 4.8.3.1 debug\_state\_t

```
enum debug_state_t
```

Defines the debug states.

**Enumerator**

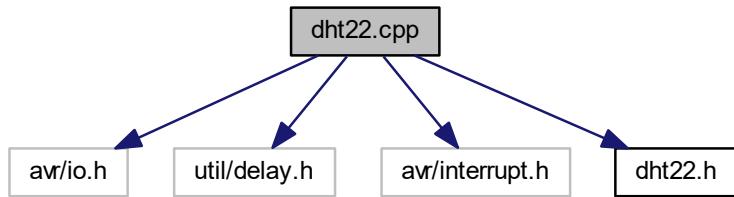
INIT	Init state : display the main menu
WAIT_INIT	Wait for a received character in init state
DISPLAY_DATA	Display sensor data in continuous
DISPLAY_CPU_LOAD	Display CPU load in continuous

Definition at line 21 of file debug.h.

## 4.9 dht22.cpp File Reference

This file defines classes for DHT22 driver.

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



### Macros

- #define MAX\_WAIT\_TIME\_US 100

#### 4.9.1 Detailed Description

This file defines classes for DHT22 driver.

#### Date

23 mars 2018

#### Author

nicls67

## 4.9.2 Macro Definition Documentation

### 4.9.2.1 MAX\_WAIT\_TIME\_US

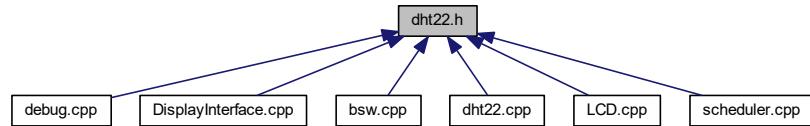
```
#define MAX_WAIT_TIME_US 100
```

Definition at line 20 of file dht22.cpp.

## 4.10 dht22.h File Reference

DHT22 driver header file.

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



### Classes

- class [dht22](#)  
*DHT 22 driver class.*

### Macros

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

### 4.10.1 Detailed Description

DHT22 driver header file.

#### Date

23 mars 2018

#### Author

nicls67

### 4.10.2 Macro Definition Documentation

#### 4.10.2.1 DHT22\_PORT

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

DHT22 is connected to port PB6

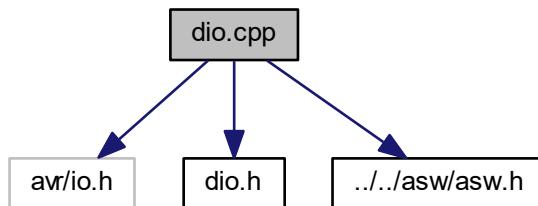
Definition at line 16 of file dht22.h.

## 4.11 dio.cpp File Reference

DIO library.

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

Include dependency graph for dio.cpp:



### 4.11.1 Detailed Description

DIO library.

Date

13 mars 2018

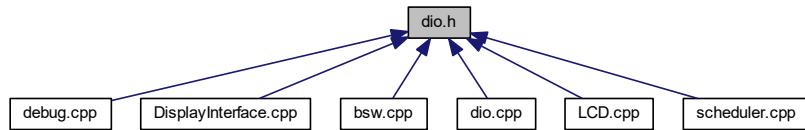
Author

nicls67

## 4.12 dio.h File Reference

DIO library header file.

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



### Classes

- class [dio](#)  
*DIO class.*

### Macros

- #define PORT\_CNF\_OUT 1
- #define PORT\_CNF\_IN 0
- #define ENCODE\_PORT(port, pin) (uint8\_t)((((uint8\_t)(port & 0xF)) << 3) | (uint8\_t)(pin & 0x7))
- #define DECODE\_PORT(portcode) (uint8\_t)((portcode >> 3) & 0xF)
- #define DECODE\_PIN(portcode) (uint8\_t)(portcode & 0x7)
- #define PORT\_A 0
- #define PORT\_B 1
- #define PORT\_C 2
- #define PORT\_D 3

### 4.12.1 Detailed Description

DIO library header file.

#### Date

13 mars 2018

#### Author

nicls67

### 4.12.2 Macro Definition Documentation

#### 4.12.2.1 DECODE\_PIN

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

Macro used to extract pin index

Definition at line 19 of file dio.h.

#### 4.12.2.2 DECODE\_PORT

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

Macro used to extract port index

Definition at line 18 of file dio.h.

#### 4.12.2.3 ENCODE\_PORT

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

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

Definition at line 17 of file dio.h.

#### 4.12.2.4 PORT\_A

```
#define PORT_A 0
```

PORTA index

Definition at line 21 of file dio.h.

#### 4.12.2.5 PORT\_B

```
#define PORT_B 1
```

PORTB index

Definition at line 22 of file dio.h.

#### 4.12.2.6 PORT\_C

```
#define PORT_C 2
```

PORTC index

Definition at line 23 of file dio.h.

#### 4.12.2.7 PORT\_CNF\_IN

```
#define PORT_CNF_IN 0
```

Pin is configured as input

Definition at line 15 of file dio.h.

#### 4.12.2.8 PORT\_CNF\_OUT

```
#define PORT_CNF_OUT 1
```

Pin is configured as output

Definition at line 14 of file dio.h.

#### 4.12.2.9 PORT\_D

```
#define PORT_D 3
```

PORTD index

Definition at line 24 of file dio.h.

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

Digital ports configuration file.

#### Macros

- #define PORTB\_CNF\_DDRB (uint8\_t)0b11000000  
*Defines the configuration of DDRB register.*
- #define PORTB\_CNF\_PORTB (uint8\_t)0b11000000  
*Defines the configuration of PORTB register.*

### 4.13.1 Detailed Description

Digital ports configuration file.

Date

19 mars 2019

Author

nicls67

### 4.13.2 Macro Definition Documentation

#### 4.13.2.1 PORTB\_CNF\_DDRB

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

Defines the configuration of DDRB register.

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

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

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

#### 4.13.2.2 PORTB\_CNF\_PORTB

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

Defines the configuration of PORTB register.

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

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

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

---

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

### Macros

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

#### 4.14.1 Macro Definition Documentation

##### 4.14.1.1 DDRA\_PTR

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

Macro defining pointer to DDR A register

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

##### 4.14.1.2 DDRB\_PTR

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

Macro defining pointer to DDR B register

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

##### 4.14.1.3 DDRC\_PTR

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

Macro defining pointer to DDR C register

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

#### 4.14.1.4 DDRD\_PTR

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

Macro defining pointer to DDR D register

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

#### 4.14.1.5 PINA\_PTR

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

Macro defining pointer to PIN A register

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

#### 4.14.1.6 PINB\_PTR

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

Macro defining pointer to PIN B register

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

#### 4.14.1.7 PINC\_PTR

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

Macro defining pointer to PIN C register

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

#### 4.14.1.8 PIND\_PTR

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

Macro defining pointer to PIN D register

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

#### 4.14.1.9 PORTA\_PTR

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

Macro defining pointer to PORT A register

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

#### 4.14.1.10 PORTB\_PTR

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

Macro defining pointer to PORT B register

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

#### 4.14.1.11 PORTC\_PTR

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

Macro defining pointer to PORT C register

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

#### 4.14.1.12 PORTD\_PTR

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

Macro defining pointer to PORT D register

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

## 4.15 DisplayInterface.cpp File Reference

Source code file for display services.

```
#include <stdlib.h>
#include <avr/io.h>
#include <util/delay.h>
#include "../../lib/LinkedList/LinkedList.h"
#include "../../scheduler/scheduler.h"
#include "../../bsw/usart/usart.h"
#include "../../bsw/timer/timer.h"
#include "../../bsw/lcd/LCD.h"
#include "../../bsw/I2C/I2C.h"
#include "../../bsw/dio/dio.h"
#include "../../bsw/dht22/dht22.h"
#include "../../bsw/cpuLoad/CpuLoad.h"
#include "../../bsw/bsw.h"
#include "../debug/debug.h"
#include "../TempSensor/TempSensor.h"
#include "../display_ift/DisplayInterface.h"
#include "../display_mgt/DisplayManagement.h"
#include "../keepAliveLed/keepAliveLed.h"
#include "../asw.h"
```

Include dependency graph for DisplayInterface.cpp:



### 4.15.1 Detailed Description

Source code file for display services.

Date

23 avr. 2019

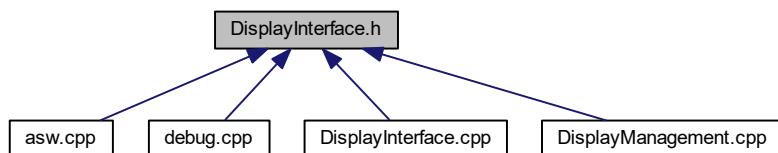
Author

nicls67

## 4.16 DisplayInterface.h File Reference

[DisplayInterface](#) class header file.

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



## Classes

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

## Macros

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

## Enumerations

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

### 4.16.1 Detailed Description

[DisplayInterface](#) class header file.

#### Date

23 avr. 2019

#### Author

nicls67

### 4.16.2 Macro Definition Documentation

#### 4.16.2.1 DISPLAY\_LINE\_SHIFT\_PERIOD\_MS

```
#define DISPLAY_LINE_SHIFT_PERIOD_MS 500
```

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

Definition at line 43 of file [DisplayInterface.h](#).

#### 4.16.2.2 DISPLAY\_LINE\_SHIFT\_TEMPO\_TIME

```
#define DISPLAY_LINE_SHIFT_TEMPO_TIME 6
```

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

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

### 4.16.3 Enumeration Type Documentation

#### 4.16.3.1 T\_DisplayInterface\_LineDisplayMode

enum [T\\_DisplayInterface\\_LineDisplayMode](#)

Modes for line display.

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

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

#### Enumerator

NORMAL	
LINE_SHIFT	
GO_TO_NEXT_LINE	

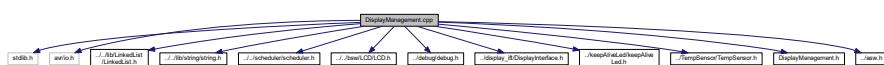
Definition at line 21 of file [DisplayInterface.h](#).

## 4.17 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/LCD/LCD.h"
#include "../debug/debug.h"
#include "../display_ift/DisplayInterface.h"
#include "../keepAliveLed/keepAliveLed.h"
#include "../TempSensor/TempSensor.h"
#include "DisplayManagement.h"
#include "../asw.h"

Include dependency graph for DisplayManagement.cpp:
```



### 4.17.1 Detailed Description

Display management source file.

**Date**

1 mai 2019

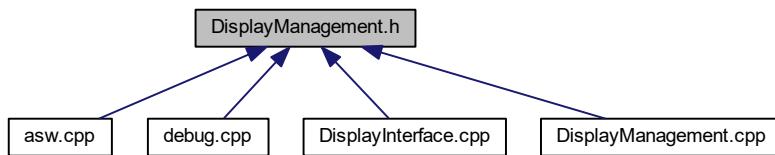
**Author**

nicls67

## 4.18 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` 5000
- #define `DISPLAY_MGT_LINE_TEMP` 0
- #define `DISPLAY_MGT_LINE_HUM` 1

### Variables

- const `T_LCD_conf_struct LCD_init_cnf`  
*LCD configuration structure.*
- const `uint8_t tempDisplayString [] = "Temperature : "`
- const `uint8_t humidityDisplayString [] = "Humidite : "`

### 4.18.1 Detailed Description

Display management class header file.

Date

1 mai 2019

Author

nicls67

### 4.18.2 Macro Definition Documentation

#### 4.18.2.1 DISPLAY\_MGT\_LCD\_I2C\_ADDR

```
#define DISPLAY_MGT_LCD_I2C_ADDR 0x27
```

Definition at line 13 of file DisplayManagement.h.

#### 4.18.2.2 DISPLAY\_MGT\_LINE\_HUM

```
#define DISPLAY_MGT_LINE_HUM 1
```

Current humidity is displayed on line 1

Definition at line 16 of file DisplayManagement.h.

#### 4.18.2.3 DISPLAY\_MGT\_LINE\_TEMP

```
#define DISPLAY_MGT_LINE_TEMP 0
```

Current temperature is displayed on line 0

Definition at line 15 of file DisplayManagement.h.

#### 4.18.2.4 DISPLAY\_MGT\_PERIOD\_TASK\_SENSOR

```
#define DISPLAY_MGT_PERIOD_TASK_SENSOR 5000
```

Display is updated every 5s

Definition at line 14 of file DisplayManagement.h.

---

### 4.18.3 Variable Documentation

#### 4.18.3.1 humidityDisplayString

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

**String** used for humidity display

Definition at line 36 of file DisplayManagement.h.

#### 4.18.3.2 LCD\_init\_cnf

```
const T_LCD_conf_struct LCD_init_cnf
```

**Initial value:**

```
= {  
    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 22 of file DisplayManagement.h.

#### 4.18.3.3 tempDisplayString

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

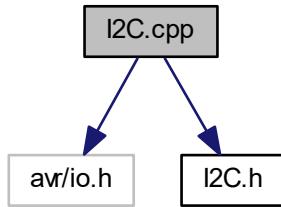
**String** used for temperature display

Definition at line 35 of file DisplayManagement.h.

## 4.19 I2C.cpp File Reference

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

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



### 4.19.1 Detailed Description

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

Date

19 avr. 2019

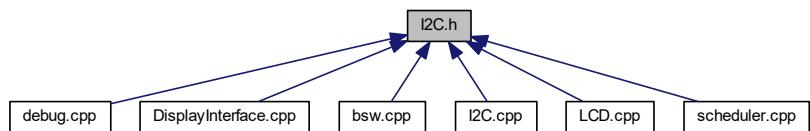
Author

nicls67

## 4.20 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

### 4.20.1 Detailed Description

[I2C](#) class header file.

#### Date

19 avr. 2019

#### Author

nicls67

### 4.20.2 Macro Definition Documentation

#### 4.20.2.1 [DATA\\_ACK](#)

```
#define DATA_ACK 0x28
```

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

Definition at line 15 of file I2C.h.

#### 4.20.2.2 [SLA\\_ACK](#)

```
#define SLA_ACK 0x18
```

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

Definition at line 14 of file I2C.h.

### 4.20.2.3 START

```
#define START 0x08

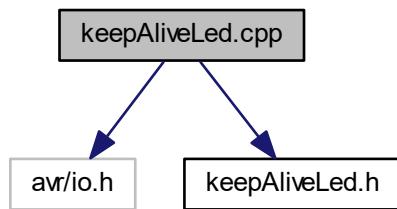
TWSR status code : START condition transmitted

Definition at line 13 of file I2C.h.
```

## 4.21 keepAliveLed.cpp File Reference

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

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



### 4.21.1 Detailed Description

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

Date

17 mars 2018

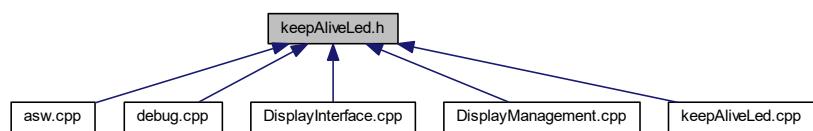
Author

nicls67

## 4.22 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)`

### 4.22.1 Detailed Description

Class [keepAliveLed](#) header file.

#### Date

17 mars 2018

#### Author

nicls67

### 4.22.2 Macro Definition Documentation

#### 4.22.2.1 LED\_PORT

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

LED is connected to port PB7

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

#### 4.22.2.2 PERIOD\_MS\_TASK\_LED

```
#define PERIOD_MS_TASK_LED SW_PERIOD_MS
```

Period for led blinking

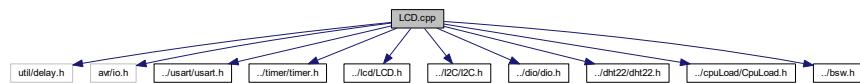
Definition at line 15 of file [keepAliveLed.h](#).

## 4.23 LCD.cpp File Reference

[LCD](#) class source file.

```
#include <util/delay.h>
#include <avr/io.h>
#include "../uart/usart.h"
#include "../timer/timer.h"
#include "../lcd/LCD.h"
#include "../I2C/I2C.h"
#include "../dio/dio.h"
#include "../dht22/dht22.h"
#include "../cpuLoad/CpuLoad.h"
#include "../bsw.h"

Include dependency graph for LCD.cpp:
```



### 4.23.1 Detailed Description

[LCD](#) class source file.

#### Date

20 avr. 2019

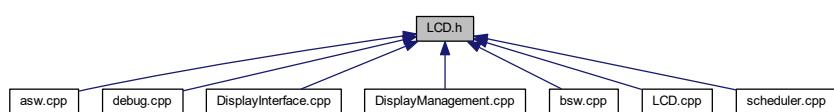
#### Author

nicls67

## 4.24 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`
- class `LCD`

*Class for `LCD S2004A` display driver.*

## Macros

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

## Enumerations

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

### 4.24.1 Detailed Description

`LCD` class header file.

#### Date

20 avr. 2019

#### Author

nicls67

### 4.24.2 Macro Definition Documentation

#### 4.24.2.1 BACKLIGHT\_PIN

```
#define BACKLIGHT_PIN 3
```

Backlight pin is on P3

Definition at line 17 of file LCD.h.

#### 4.24.2.2 EN\_PIN

```
#define EN_PIN 2
```

EN bit is on P2

Definition at line 14 of file LCD.h.

#### 4.24.2.3 LCD\_CNF\_BACKLIGHT\_OFF

```
#define LCD_CNF_BACKLIGHT_OFF 0
```

Backlight is disabled

Definition at line 70 of file LCD.h.

#### 4.24.2.4 LCD\_CNF\_BACKLIGHT\_ON

```
#define LCD_CNF_BACKLIGHT_ON 1
```

Backlight is enabled

Definition at line 69 of file LCD.h.

#### 4.24.2.5 LCD\_CNF\_CURSOR\_BLINK\_OFF

```
#define LCD_CNF_CURSOR_BLINK_OFF 0
```

Cursor blinking is off, bit is set to 0

Definition at line 58 of file LCD.h.

#### 4.24.2.6 LCD\_CNF\_CURSOR\_BLINK\_ON

```
#define LCD_CNF_CURSOR_BLINK_ON 1
```

Cursor blinking is on, bit is set to 1

Definition at line 57 of file LCD.h.

#### 4.24.2.7 LCD\_CNF\_CURSOR\_OFF

```
#define LCD_CNF_CURSOR_OFF 0
```

Cursor is off, bit is set to 0

Definition at line 54 of file LCD.h.

#### 4.24.2.8 LCD\_CNF\_CURSOR\_ON

```
#define LCD_CNF_CURSOR_ON 1
```

Cursor is on, bit is set to 1

Definition at line 53 of file LCD.h.

#### 4.24.2.9 LCD\_CNF\_DISPLAY\_OFF

```
#define LCD_CNF_DISPLAY_OFF 0
```

Display is off, bit is set to 0

Definition at line 50 of file LCD.h.

#### 4.24.2.10 LCD\_CNF\_DISPLAY\_ON

```
#define LCD_CNF_DISPLAY_ON 1
```

Display is on, bit is set to 1

Definition at line 49 of file LCD.h.

#### 4.24.2.11 LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_LEFT

```
#define LCD_CNF_ENTRY_MODE_DIRECTION_LEFT 0
```

Direction of shift is left, bit is set to 0

Definition at line 62 of file LCD.h.

#### 4.24.2.12 LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_RIGHT

```
#define LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT 1
```

Direction of shift is right, bit is set to 1

Definition at line 61 of file LCD.h.

#### 4.24.2.13 LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT\_OFF

```
#define LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF 0
```

Display shift is not performed, bit is set to 0

Definition at line 66 of file LCD.h.

#### 4.24.2.14 LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT\_ON

```
#define LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON 1
```

Display shift is performed, bit is set to 1

Definition at line 65 of file LCD.h.

#### 4.24.2.15 LCD\_CNF\_FONT\_5\_11

```
#define LCD_CNF_FONT_5_11 1
```

Two-line configuration, bit is set to 1

Definition at line 46 of file LCD.h.

#### 4.24.2.16 LCD\_CNF\_FONT\_5\_8

```
#define LCD_CNF_FONT_5_8 0
```

One-line configuration, bit is set to 0

Definition at line 45 of file LCD.h.

#### 4.24.2.17 LCD\_CNF\_ONE\_LINE

```
#define LCD_CNF_ONE_LINE 0
```

One-line configuration, bit is set to 0

Definition at line 41 of file LCD.h.

**4.24.2.18 LCD\_CNF\_SHIFT\_ID**

```
#define LCD_CNF_SHIFT_ID 1
```

Field ID (increment or decrement) of command "entry mode set" is on bit DB1

Definition at line 37 of file LCD.h.

**4.24.2.19 LCD\_CNF\_SHIFT\_SH**

```
#define LCD_CNF_SHIFT_SH 0
```

Field SH (shift of display) of command "entry mode set" is on bit DB1

Definition at line 38 of file LCD.h.

**4.24.2.20 LCD\_CNF\_TWO\_LINE**

```
#define LCD_CNF_TWO_LINE 1
```

Two-line configuration, bit is set to 1

Definition at line 42 of file LCD.h.

**4.24.2.21 LCD\_DISPLAY\_CTRL\_FIELD\_B**

```
#define LCD_DISPLAY_CTRL_FIELD_B 0
```

Field B (cursor blink) of command "display control" is on bit DB0

Definition at line 34 of file LCD.h.

**4.24.2.22 LCD\_DISPLAY\_CTRL\_FIELD\_C**

```
#define LCD_DISPLAY_CTRL_FIELD_C 1
```

Field C (cursor on/off) of command "display control" is on bit DB1

Definition at line 33 of file LCD.h.

#### 4.24.2.23 LCD\_DISPLAY\_CTRL\_FIELD\_D

```
#define LCD_DISPLAY_CTRL_FIELD_D 2
```

Field D (display on/off) of command "display control" is on bit DB2

Definition at line 32 of file LCD.h.

#### 4.24.2.24 LCD\_FCT\_SET\_FIELD\_DL

```
#define LCD_FCT_SET_FIELD_DL 4
```

Field DL (data length) of command "function set" is on bit DB4

Definition at line 27 of file LCD.h.

#### 4.24.2.25 LCD\_FCT\_SET\_FIELD\_F

```
#define LCD_FCT_SET_FIELD_F 2
```

Field F (font type) of command "function set" is on bit DB2

Definition at line 29 of file LCD.h.

#### 4.24.2.26 LCD\_FCT\_SET\_FIELD\_N

```
#define LCD_FCT_SET_FIELD_N 3
```

Field N (number of lines) of command "function set" is on bit DB3

Definition at line 28 of file LCD.h.

#### 4.24.2.27 LCD\_INST\_CLR\_DISPLAY\_BIT

```
#define LCD_INST_CLR_DISPLAY_BIT 0
```

Instruction bit for "clear display" is DB0

Definition at line 20 of file LCD.h.

**4.24.2.28 LCD\_INST\_DISPLAY\_CTRL**

```
#define LCD_INST_DISPLAY_CTRL 3
```

Instruction bit for "display control" is DB3

Definition at line 22 of file LCD.h.

**4.24.2.29 LCD\_INST\_ENTRY\_MODE\_SET**

```
#define LCD_INST_ENTRY_MODE_SET 2
```

Instruction bit for "entry mode" is DB2

Definition at line 23 of file LCD.h.

**4.24.2.30 LCD\_INST\_FUNCTION\_SET**

```
#define LCD_INST_FUNCTION_SET 5
```

Instruction bit for "function set" is DB5

Definition at line 21 of file LCD.h.

**4.24.2.31 LCD\_INST\_SET\_DDRAM\_ADDR**

```
#define LCD_INST_SET_DDRAM_ADDR 7
```

Instruction bit for "set DDRAM address" is DB7

Definition at line 24 of file LCD.h.

**4.24.2.32 LCD\_RAM\_1\_LINE\_MAX**

```
#define LCD_RAM_1_LINE_MAX 0x4F
```

Maximum address value in 1-line mode

Definition at line 74 of file LCD.h.

#### 4.24.2.33 LCD\_RAM\_1\_LINE\_MIN

```
#define LCD_RAM_1_LINE_MIN 0
```

Minimum address value in 1-line mode

Definition at line 73 of file LCD.h.

#### 4.24.2.34 LCD\_RAM\_2\_LINES\_MAX\_1

```
#define LCD_RAM_2_LINES_MAX_1 0x27
```

Maximum address value in 2-lines mode for line 1

Definition at line 76 of file LCD.h.

#### 4.24.2.35 LCD\_RAM\_2\_LINES\_MAX\_2

```
#define LCD_RAM_2_LINES_MAX_2 0x67
```

Maximum address value in 2-lines mode for line 2

Definition at line 78 of file LCD.h.

#### 4.24.2.36 LCD\_RAM\_2\_LINES\_MIN\_1

```
#define LCD_RAM_2_LINES_MIN_1 0
```

Minimum address value in 2-lines mode for line 1

Definition at line 75 of file LCD.h.

#### 4.24.2.37 LCD\_RAM\_2\_LINES\_MIN\_2

```
#define LCD_RAM_2_LINES_MIN_2 0x40
```

Minimum address value in 2-lines mode for line 2

Definition at line 77 of file LCD.h.

#### 4.24.2.38 LCD\_SIZE\_NB\_CHAR\_PER\_LINE

```
#define LCD_SIZE_NB_CHAR_PER_LINE 20
```

LCD screen has 20 characters per line

Definition at line 85 of file LCD.h.

#### 4.24.2.39 LCD\_SIZE\_NB\_LINES

```
#define LCD_SIZE_NB_LINES 4
```

LCD screen has 4 lines

Definition at line 86 of file LCD.h.

#### 4.24.2.40 LCD\_WAIT\_CLR\_RETURN

```
#define LCD_WAIT_CLR_RETURN 1600
```

Waiting time after clear display and return home operations is at least 1520 us

Definition at line 81 of file LCD.h.

#### 4.24.2.41 LCD\_WAIT\_OTHER\_MODES

```
#define LCD_WAIT_OTHER_MODES 40
```

Waiting time after all other modes is at least 38 us

Definition at line 82 of file LCD.h.

#### 4.24.2.42 RS\_PIN

```
#define RS_PIN 0
```

RS pin is on P0

Definition at line 16 of file LCD.h.

#### 4.24.2.43 RW\_PIN

```
#define RW_PIN 1
```

RW pin is on P1

Definition at line 15 of file LCD.h.

### 4.24.3 Enumeration Type Documentation

#### 4.24.3.1 T\_LCD\_command

```
enum T_LCD_command
```

LCD commands enumeration.

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

Enumerator

LCD_CMD_FUNCTION_SET	
LCD_CMD_CLEAR_DISPLAY	
LCD_CMD_DISPLAY_CTRL	
LCD_CMD_ENTRY_MODE_SET	
LCD_CMD_SET_DDRAM_ADDR	

Definition at line 93 of file LCD.h.

#### 4.24.3.2 T\_LCD\_config\_mode

```
enum T_LCD_config_mode
```

LCD modes enumeration.

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

Enumerator

LCD_MODE_INSTRUCTION	
LCD_MODE_DATA	

Definition at line 107 of file LCD.h.

#### 4.24.3.3 T\_LCD\_ram\_area

enum [T\\_LCD\\_ram\\_area](#)

Screen RAM definition.

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

Enumerator

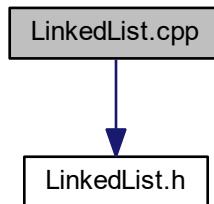
LCD_DATA_DDRAM	
LCD_DATA_CGRAM	

Definition at line 118 of file LCD.h.

## 4.25 LinkedList.cpp File Reference

#include "LinkedList.h"

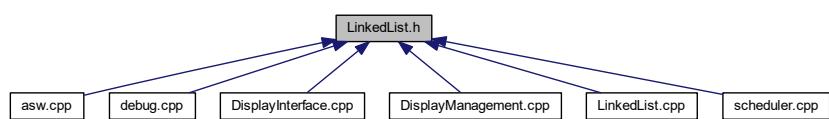
Include dependency graph for LinkedList.cpp:



## 4.26 LinkedList.h File Reference

Linked List library header file.

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



## Classes

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

## TypeDefs

- [typedef bool\(\\* CompareFctPtr\\_t\) \(void \\*LLElement, void \\*CompareElement\)](#)

### 4.26.1 Detailed Description

Linked List library header file.

#### Date

27 avr. 2019

#### Author

nicls67

### 4.26.2 Typedef Documentation

#### 4.26.2.1 CompareFctPtr\_t

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

Type defining a pointer to the comparison function

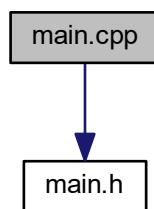
Definition at line 14 of file `LinkedList.h`.

## 4.27 main.cpp File Reference

Background task file.

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

Include dependency graph for `main.cpp`:



## Functions

- **ISR (TIMER1\_COMPA\_vect)**  
*Main software interrupt.*
- **ISR (USART0\_RX\_vect)**  
*USART Rx Complete interrupt.*
- int **main (void)**  
*Background task of program.*

### 4.27.1 Detailed Description

Background task file.

#### Date

12 mars 2018

#### Author

nicls67

### 4.27.2 Function Documentation

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

```
ISR (
    TIMER1_COMPA_vect )
```

Main software interrupt.

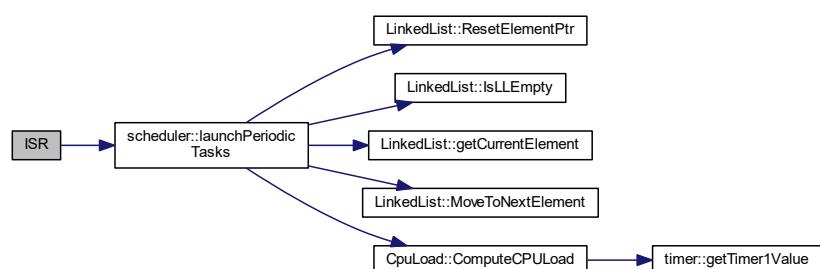
This function handles the interrupt raised by Timer #1. It wakes up the software every 500 ms to perform applications.

#### Returns

Nothing

Definition at line 19 of file main.cpp.

Here is the call graph for this function:



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

```
ISR (
    USART0_RX_vect
)
```

USART Rx Complete interrupt.

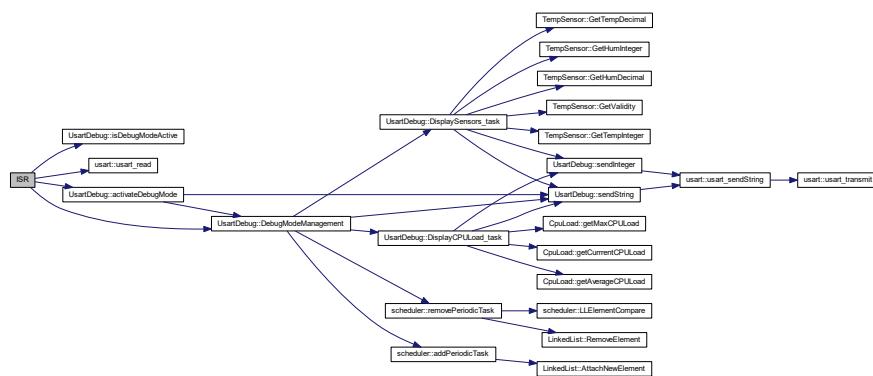
This function handles the interrupt raised when a frame has been received by USART. If debug mode mode is active, it calls debug mode management function. If inactive, it calls debug mode activation function if the received character is 'a'

Returns

Nothing

Definition at line 31 of file main.cpp.

Here is the call graph for this function:



#### 4.27.2.3 main()

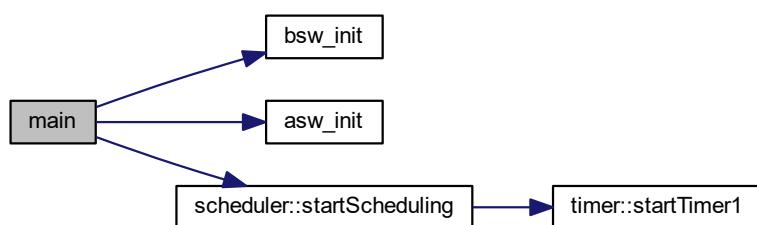
```
int main (
    void
)
```

Background task of program.

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

Definition at line 51 of file main.cpp.

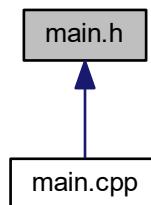
Here is the call graph for this function:



## 4.28 main.h File Reference

Background task header file.

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



### 4.28.1 Detailed Description

Background task header file.

Date

17 mars 2018

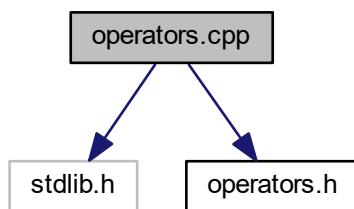
Author

nicls67

## 4.29 operators.cpp File Reference

c++ operators definitions

```
#include <stdlib.h>
#include "operators.h"
Include dependency graph for operators.cpp:
```



## Functions

- `void * operator new (size_t a_size)`  
*Operator new.*
- `void operator delete (void *ptr)`  
*Operator delete.*

### 4.29.1 Detailed Description

c++ operators definitions

#### Date

14 mars 2018

#### Author

nicls67

### 4.29.2 Function Documentation

#### 4.29.2.1 operator delete()

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

Operator delete.

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

#### Parameters

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

#### Returns

Nothing

Definition at line 18 of file operators.cpp.

#### 4.29.2.2 operator new()

```
void* operator new (
    size_t a_size )
```

Operator new.

Equivalent to malloc function in C Allocates a memory zone of size a\_size

**Parameters**

in	a_size	memory size to allocate
----	--------	-------------------------

**Returns**

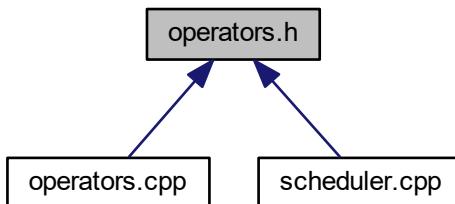
Pointer to the start of allocated memory zone

Definition at line 13 of file operators.cpp.

## 4.30 operators.h File Reference

c++ operators definitions header file

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



### Functions

- void \* **operator new** (size\_t a\_size)  
*Operator new.*
- void **operator delete** (void \*ptr)  
*Operator delete.*

#### 4.30.1 Detailed Description

c++ operators definitions header file

**Date**

14 mars 2018

**Author**

nicls67

## 4.30.2 Function Documentation

### 4.30.2.1 operator delete()

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

Operator delete.

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

#### Parameters

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

#### Returns

Nothing

Definition at line 18 of file operators.cpp.

### 4.30.2.2 operator new()

```
void* operator new (
    size_t a_size )
```

Operator new.

Equivalent to malloc function in C Allocates a memory zone of size a\_size

#### Parameters

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

#### Returns

Pointer to the start of allocated memory zone

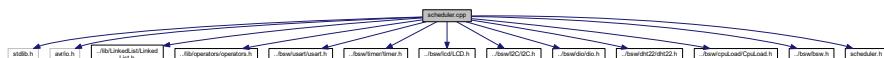
Definition at line 13 of file operators.cpp.

## 4.31 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/usart/usart.h"
#include "../bsw/timer/timer.h"
#include "../bsw/lcd/LCD.h"
#include "../bsw/I2C/I2C.h"
#include "../bsw/dio/dio.h"
#include "../bsw/dht22/dht22.h"
#include "../bsw/cpuLoad/CpuLoad.h"
#include "../bsw/bsw.h"
#include "scheduler.h"

Include dependency graph for scheduler.cpp:
```



## Variables

- `scheduler * p_scheduler`

### 4.31.1 Detailed Description

Defines scheduler class.

#### Date

16 mars 2018

#### Author

nicls67

### 4.31.2 Variable Documentation

#### 4.31.2.1 p\_scheduler

`scheduler* p_scheduler`

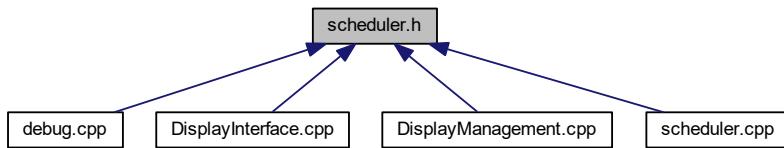
Pointer to scheduler object

Definition at line 28 of file scheduler.cpp.

## 4.32 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_scheduler`

#### 4.32.1 Detailed Description

Scheduler class header file.

##### Date

16 mars 2018

##### Author

nicls67

## 4.32.2 Macro Definition Documentation

### 4.32.2.1 PRESCALER\_PERIODIC\_TIMER

```
#define PRESCALER_PERIODIC_TIMER 256
```

Value of prescaler to use for periodic timer

Definition at line 16 of file scheduler.h.

### 4.32.2.2 SW\_PERIOD\_MS

```
#define SW_PERIOD_MS 500
```

Software period, used to define periodic timer interrupt

Definition at line 15 of file scheduler.h.

### 4.32.2.3 TIMER\_CTC\_VALUE

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

Compare value for periodic timer

Definition at line 17 of file scheduler.h.

## 4.32.3 Typedef Documentation

### 4.32.3.1 TaskPtr\_t

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

Type defining a pointer to function.

Definition at line 22 of file scheduler.h.

## 4.32.4 Variable Documentation

#### 4.32.4.1 p\_scheduler

```
scheduler* p_scheduler
```

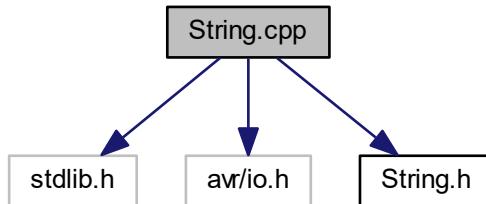
Pointer to scheduler object

Definition at line 28 of file scheduler.cpp.

## 4.33 String.cpp File Reference

[String](#) class source file.

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



### 4.33.1 Detailed Description

[String](#) class source file.

#### Date

2 mai 2019

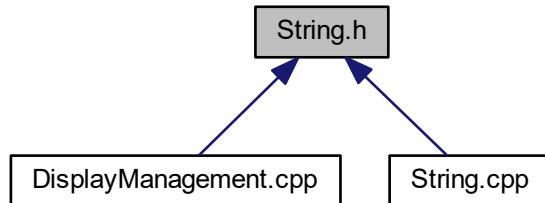
#### Author

nicls67

## 4.34 String.h File Reference

[String](#) class header file.

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



### Classes

- class [String](#)  
*String management class.*

#### 4.34.1 Detailed Description

[String](#) class header file.

##### Date

2 mai 2019

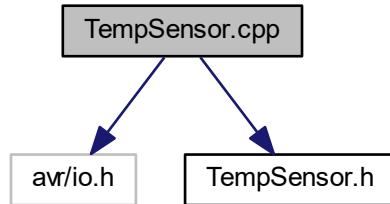
##### Author

nicls67

## 4.35 TempSensor.cpp File Reference

Defines function of class [TempSensor](#).

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



## Macros

- `#define PIT_BEFORE_INVALID 60`

### 4.35.1 Detailed Description

Defines function of class [TempSensor](#).

#### Date

23 mars 2018

#### Author

nicls67

### 4.35.2 Macro Definition Documentation

#### 4.35.2.1 PIT\_BEFORE\_INVALID

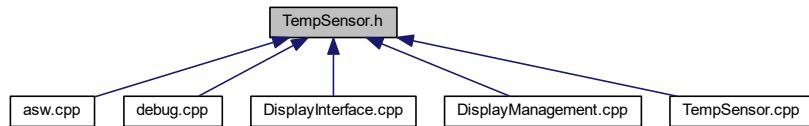
```
#define PIT_BEFORE_INVALID 60
```

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

## 4.36 TempSensor.h File Reference

Class [TempSensor](#) header file.

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



### Classes

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

### Macros

- `#define PERIOD_MS_TASK_TEMP_SENSOR 5000`

#### 4.36.1 Detailed Description

Class [TempSensor](#) header file.

##### Date

23 mars 2018

##### Author

nicls67

#### 4.36.2 Macro Definition Documentation

##### 4.36.2.1 PERIOD\_MS\_TASK\_TEMP\_SENSOR

```
#define PERIOD_MS_TASK_TEMP_SENSOR 5000
```

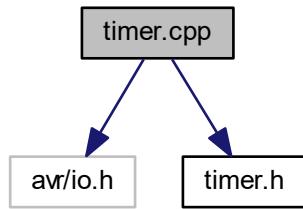
Period for reading temperature data

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

## 4.37 timer.cpp File Reference

Defines function for class timer.

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



### 4.37.1 Detailed Description

Defines function for class timer.

#### Date

15 mars 2018

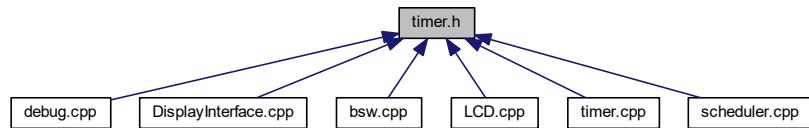
#### Author

nicls67

## 4.38 timer.h File Reference

Timer class header file.

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



## Classes

- class [timer](#)

*Class defining a timer.*

### 4.38.1 Detailed Description

Timer class header file.

#### Date

15 mars 2018

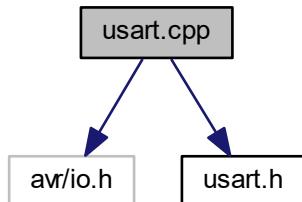
#### Author

nicls67

## 4.39 usart.cpp File Reference

BSW library for USART.

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



### 4.39.1 Detailed Description

BSW library for USART.

#### Date

13 mars 2018

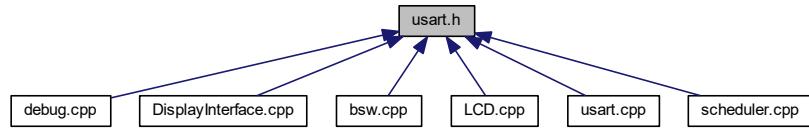
#### Author

nicls67

## 4.40 usart.h File Reference

Header file for USART library.

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



### Classes

- class [uart](#)

*USART serial bus class.*

#### 4.40.1 Detailed Description

Header file for USART library.

##### Date

13 mars 2018

##### Author

nicls67



# Index

ASW\_cnf\_struct  
  asw.cpp, 116  
  asw.h, 118  
activateDebugMode  
  UsartDebug, 107  
addPeriodicTask  
  scheduler, 65  
appendInteger  
  String, 72  
appendString  
  String, 73  
asw.cpp, 115  
  ASW\_cnf\_struct, 116  
  asw\_init, 116  
asw.h, 117  
  ASW\_cnf\_struct, 118  
  asw\_init, 118  
asw\_init  
  asw.cpp, 116  
  asw.h, 118  
AttachNewElement  
  LinkedList, 59  
avg\_load  
  CpuLoad, 8  
BACKLIGHT\_PIN  
  LCD.h, 149  
BSW\_cnf\_struct  
  bsw.cpp, 120  
  bsw.h, 122  
backlight\_en  
  T\_LCD\_conf\_struct, 84  
backlight\_enable  
  LCD, 55  
BaudRate  
  usart, 105  
bitrate  
  I2C, 38  
blinkLed\_task  
  keepAliveLed, 39  
bsw.cpp, 119  
  BSW\_cnf\_struct, 120  
  bsw\_init, 119  
bsw.h, 120  
  BSW\_cnf\_struct, 122  
  bsw\_init, 122  
  I2C\_BITRATE, 121  
  USART\_BAUDRATE, 121  
bsw\_init  
  bsw.cpp, 119  
bsw.h, 122  
  ASW\_cnf\_struct, 122  
  bsw\_init, 122  
  LCD, 56  
  cnfCursorBlink  
    LCD, 55  
  cnfCursorOnOff  
    LCD, 55  
  cnfDisplayOnOff  
    LCD, 55  
  cnfEntryModeDir  
    LCD, 56  
  cnfEntryModeShift  
    LCD, 56  
  cnfFontType  
    LCD, 56  
  cnfI2C\_addr  
    LCD, 56  
  cnfLineNumber  
    LCD, 56  
  command  
    LCD, 43  
  CompareFctPtr\_t  
    LinkedList.h, 160  
  ComputeCPULoad  
    CpuLoad, 6  
  ComputeStringSize  
    String, 74  
  ConfigureBacklight  
    LCD, 43  
  ConfigureCursorBlink  
    LCD, 44  
  ConfigureCursorOnOff  
    LCD, 45  
  ConfigureDisplayOnOff  
    LCD, 45  
  ConfigureEntryModeDir  
    LCD, 46  
  ConfigureEntryModeShift  
    LCD, 47  
  ConfigureFontType  
    LCD, 47  
  ConfigureI2CAddr  
    LCD, 48  
  ConfigureLineNumber  
    LCD, 49  
  configureTimer1  
    timer, 98

CpuLoad, 5  
 avg\_load, 8  
 ComputeCPULoad, 6  
 CpuLoad, 6  
 current\_load, 8  
 getAverageCPULoad, 6  
 getCurrentCPULoad, 7  
 getMaxCPULoad, 7  
 last\_sum\_value, 8  
 max\_load, 9  
 sample\_cnt, 9  
 sample\_idx, 9  
 sample\_mem, 9  
 CpuLoad.cpp, 123  
 CpuLoad.h, 123  
 NB\_OF\_SAMPLES, 124  
 curElement\_ptr  
 LinkedList, 62  
 current\_load  
 CpuLoad, 8  
 cursor\_en  
 T\_LCD\_conf\_struct, 84  
 cursorBlink\_en  
 T\_LCD\_conf\_struct, 84  
 DATA\_ACK  
 I2C.h, 144  
 DDRA\_PTR  
 dio\_reg\_atm2560.h, 134  
 DDRB\_PTR  
 dio\_reg\_atm2560.h, 134  
 DDRC\_PTR  
 dio\_reg\_atm2560.h, 134  
 DDRD\_PTR  
 dio\_reg\_atm2560.h, 134  
 DECODE\_PIN  
 dio.h, 130  
 DECODE\_PORT  
 dio.h, 131  
 DHT22\_PORT  
 dht22.h, 129  
 DISPLAY\_LINE\_SHIFT\_PERIOD\_MS  
 DisplayInterface.h, 138  
 DISPLAY\_LINE\_SHIFT\_TEMPO\_TIME  
 DisplayInterface.h, 138  
 DISPLAY\_MGT\_LCD\_I2C\_ADDR  
 DisplayManagement.h, 141  
 DISPLAY\_MGT\_LINE\_HUM  
 DisplayManagement.h, 141  
 DISPLAY\_MGT\_LINE\_TEMP  
 DisplayManagement.h, 141  
 DISPLAY\_MGT\_PERIOD\_TASK\_SENSOR  
 DisplayManagement.h, 141  
 data\_ptr  
 LinkedList::T\_LL\_element, 86  
 ddram\_addr  
 LCD, 57  
 debug.cpp, 124  
 str\_debug\_main\_menu, 125  
 debug.h, 125  
 debug\_state\_t, 126  
 PERIOD\_MS\_TASK\_DISPLAY\_CPU\_LOAD, 126  
 PERIOD\_MS\_TASK\_DISPLAY\_SENSORS, 126  
 debug\_state  
 UsartDebug, 113  
 debug\_state\_t  
 debug.h, 126  
 debugModeActive\_F  
 UsartDebug, 113  
 DebugModeManagement  
 UsartDebug, 107  
 dht22, 10  
 dht22, 10  
 initializeCommunication, 11  
 read, 11  
 dht22.cpp, 127  
 MAX\_WAIT\_TIME\_US, 128  
 dht22.h, 128  
 DHT22\_PORT, 129  
 dio, 12  
 dio, 13  
 dio\_changePortPinCnf, 14  
 dio\_getPort, 15  
 dio\_getPort\_fast, 16  
 dio\_invertPort, 16  
 dio\_memorizePINaddress, 17  
 dio\_setPort, 18  
 getDDRxAddress, 19  
 getPINxAddress, 19  
 getPORTxAddress, 20  
 PINx\_addr\_mem, 21  
 PINx\_idx\_mem, 21  
 ports\_init, 21  
 dio.cpp, 129  
 dio.h, 130  
 DECODE\_PIN, 130  
 DECODE\_PORT, 131  
 ENCODE\_PORT, 131  
 PORT\_CNF\_IN, 132  
 PORT\_CNF\_OUT, 132  
 PORT\_A, 131  
 PORT\_B, 131  
 PORT\_C, 131  
 PORT\_D, 132  
 dio\_changePortPinCnf  
 dio, 14  
 dio\_getPort  
 dio, 15  
 dio\_getPort\_fast  
 dio, 16  
 dio\_invertPort  
 dio, 16  
 dio\_memorizePINaddress  
 dio, 17  
 dio\_port\_cnf.h, 132  
 PORTB\_CNF\_DDRB, 133  
 PORTB\_CNF\_PORTB, 133

dio\_reg\_atm2560.h, 134  
DDRA\_PTR, 134  
DDRB\_PTR, 134  
DDRC\_PTR, 134  
DDRD\_PTR, 134  
PINA\_PTR, 135  
PINB\_PTR, 135  
PINC\_PTR, 135  
PIND\_PTR, 135  
PORTA\_PTR, 135  
PORTB\_PTR, 136  
PORTC\_PTR, 136  
PORTD\_PTR, 136  
dio\_setPort  
    dio, 18  
display\_en  
    T\_LCD\_conf\_struct, 84  
DisplayCPUload\_task  
    UsartDebug, 109  
DisplayFullLine  
    DisplayInterface, 24  
DisplayInterface, 22  
    ClearLine, 24  
    DisplayFullLine, 24  
    DisplayInterface, 23  
    dummy, 29  
    FindFirstCharAddr, 25  
    getLLShiftDataPtr, 26  
    IsLineEmpty, 27  
    LL\_shift\_data\_ptr, 29  
    LLElementCompare, 27  
    lineEmptyTab, 29  
    p\_lcd, 29  
    shiftLine\_task, 28  
DisplayInterface.cpp, 137  
DisplayInterface.h, 137  
    DISPLAY\_LINE\_SHIFT\_PERIOD\_MS, 138  
    DISPLAY\_LINE\_SHIFT\_TEMPO\_TIME, 138  
    T\_DisplayInterface\_LineDisplayMode, 139  
DisplayManagement, 30  
    DisplayManagement, 31  
    DisplaySensorData\_Task, 31  
    GetIftPointer, 32  
    GetTempSensorPtr, 33  
    p\_display\_ift, 34  
    p\_tempSensor, 34  
DisplayManagement.cpp, 139  
DisplayManagement.h, 140  
    DISPLAY\_MGT\_LCD\_I2C\_ADDR, 141  
    DISPLAY\_MGT\_LINE\_HUM, 141  
    DISPLAY\_MGT\_LINE\_TEMP, 141  
    DISPLAY\_MGT\_PERIOD\_TASK\_SENSOR, 141  
    humidityDisplayString, 142  
    LCD\_init\_cnf, 142  
    tempDisplayString, 142  
DisplaySensorData\_Task  
    DisplayManagement, 31  
DisplaySensors\_task  
    UsartDebug, 109  
    dummy  
        DisplayInterface, 29  
    EN\_PIN  
        LCD.h, 149  
    ENCODE\_PORT  
        dio.h, 131  
    entryModeDir  
        T\_LCD\_conf\_struct, 84  
    entryModeShift  
        T\_LCD\_conf\_struct, 84  
    FindFirstCharAddr  
        DisplayInterface, 25  
    firstElement  
        LinkedList, 63  
    fontType\_cnf  
        T\_LCD\_conf\_struct, 85  
    getAverageCPUload  
        CpuLoad, 6  
    getCurrentElement  
        LinkedList, 60  
    getCurrentCPUload  
        CpuLoad, 7  
    GetDDRAMAddress  
        LCD, 49  
    getDDRxAddress  
        dio, 19  
    GetHumDecimal  
        TempSensor, 89  
    GetHumInteger  
        TempSensor, 90  
    getHumPtr  
        TempSensor, 90  
    getHumidity  
        TempSensor, 89  
    GetIftPointer  
        DisplayManagement, 32  
    getLLShiftDataPtr  
        DisplayInterface, 26  
    GetLineNumberCnf  
        LCD, 50  
    getMaxCPUload  
        CpuLoad, 7  
    getPINxAddress  
        dio, 19  
    getPORTxAddress  
        dio, 20  
    getPitNumber  
        scheduler, 66  
    getSize  
        String, 76  
    getString  
        String, 76  
    getTemp  
        TempSensor, 91  
    GetTempDecimal

TempSensor, 91  
 GetTempInteger  
     TempSensor, 92  
 getTempPtr  
     TempSensor, 92  
 GetTempSensorPtr  
     DisplayManagement, 33  
 getTimer1Value  
     timer, 99  
 GetValidity  
     TempSensor, 93  
 humidityDisplayString  
     DisplayManagement.h, 142  
 I2C.cpp, 143  
 I2C.h, 143  
     DATA\_ACK, 144  
     SLA\_ACK, 144  
     START, 144  
 I2C\_BITRATE  
     bsw.h, 121  
 I2C, 34  
     bitrate, 38  
     I2C, 35  
     initializeBus, 36  
     setBitRate, 36  
     setTxAddress, 37  
     tx\_address, 38  
     writeByte, 37  
 i2c\_addr  
     T\_LCD\_conf\_struct, 85  
 ISR  
     main.cpp, 161  
 initializeBus  
     I2C, 36  
 initializeCommunication  
     dht22, 11  
 InitializeScreen  
     LCD, 50  
 isDebugModeActive  
     UsartDebug, 110  
 IsLLEmpty  
     LinkedList, 60  
 IsLineEmpty  
     DisplayInterface, 27  
 keepAliveLed, 38  
     blinkLed\_task, 39  
     keepAliveLed, 39  
 keepAliveLed.cpp, 145  
 keepAliveLed.h, 145  
     LED\_PORT, 146  
     PERIOD\_MS\_TASK\_LED, 146  
 LCD.cpp, 147  
 LCD.h, 147  
     BACKLIGHT\_PIN, 149  
     EN\_PIN, 149  
     LCD\_CNF\_BACKLIGHT\_OFF, 149  
     LCD\_CNF\_BACKLIGHT\_ON, 150  
     LCD\_CNF\_CURSOR\_BLINK\_OFF, 150  
     LCD\_CNF\_CURSOR\_BLINK\_ON, 150  
     LCD\_CNF\_CURSOR\_OFF, 150  
     LCD\_CNF\_CURSOR\_ON, 150  
     LCD\_CNF\_DISPLAY\_OFF, 151  
     LCD\_CNF\_DISPLAY\_ON, 151  
     LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_LEFT, 151  
     LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_RIGHT, 151  
     LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT←OFF, 151  
     LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT←ON, 152  
     LCD\_CNF\_FONT\_5\_11, 152  
     LCD\_CNF\_FONT\_5\_8, 152  
     LCD\_CNF\_ONE\_LINE, 152  
     LCD\_CNF\_SHIFT\_ID, 152  
     LCD\_CNF\_SHIFT\_SH, 153  
     LCD\_CNF\_TWO\_LINE, 153  
     LCD\_DISPLAY\_CTRL\_FIELD\_B, 153  
     LCD\_DISPLAY\_CTRL\_FIELD\_C, 153  
     LCD\_DISPLAY\_CTRL\_FIELD\_D, 153  
     LCD\_FCT\_SET\_FIELD\_DL, 154  
     LCD\_FCT\_SET\_FIELD\_F, 154  
     LCD\_FCT\_SET\_FIELD\_N, 154  
     LCD\_INST\_CLR\_DISPLAY\_BIT, 154  
     LCD\_INST\_DISPLAY\_CTRL, 154  
     LCD\_INST\_ENTRY\_MODE\_SET, 155  
     LCD\_INST\_FUNCTION\_SET, 155  
     LCD\_INST\_SET\_DDRAM\_ADDR, 155  
     LCD\_RAM\_1\_LINE\_MAX, 155  
     LCD\_RAM\_1\_LINE\_MIN, 155  
     LCD\_RAM\_2\_LINES\_MAX\_1, 156  
     LCD\_RAM\_2\_LINES\_MAX\_2, 156  
     LCD\_RAM\_2\_LINES\_MIN\_1, 156  
     LCD\_RAM\_2\_LINES\_MIN\_2, 156  
     LCD\_SIZE\_NB\_CHAR\_PER\_LINE, 156  
     LCD\_SIZE\_NB\_LINES, 157  
     LCD\_WAIT\_CLR\_RETURN, 157  
     LCD\_WAIT\_OTHER\_MODES, 157  
     RS\_PIN, 157  
     RW\_PIN, 157  
     T\_LCD\_command, 158  
     T\_LCD\_config\_mode, 158  
     T\_LCD\_ram\_area, 158  
     LCD\_CNF\_BACKLIGHT\_OFF  
         LCD.h, 149  
     LCD\_CNF\_BACKLIGHT\_ON  
         LCD.h, 150  
     LCD\_CNF\_CURSOR\_BLINK\_OFF  
         LCD.h, 150  
     LCD\_CNF\_CURSOR\_BLINK\_ON  
         LCD.h, 150  
     LCD\_CNF\_CURSOR\_OFF  
         LCD.h, 150

LCD\_CNF\_CURSOR\_ON  
LCD.h, 150

LCD\_CNF\_DISPLAY\_OFF  
LCD.h, 151

LCD\_CNF\_DISPLAY\_ON  
LCD.h, 151

LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_LEFT  
LCD.h, 151

LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_RIGHT  
LCD.h, 151

LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT\_OFF  
LCD.h, 151

LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT\_ON  
LCD.h, 152

LCD\_CNF\_FONT\_5\_11  
LCD.h, 152

LCD\_CNF\_FONT\_5\_8  
LCD.h, 152

LCD\_CNF\_ONE\_LINE  
LCD.h, 152

LCD\_CNF\_SHIFT\_ID  
LCD.h, 152

LCD\_CNF\_SHIFT\_SH  
LCD.h, 153

LCD\_CNF\_TWO\_LINE  
LCD.h, 153

LCD\_DISPLAY\_CTRL\_FIELD\_B  
LCD.h, 153

LCD\_DISPLAY\_CTRL\_FIELD\_C  
LCD.h, 153

LCD\_DISPLAY\_CTRL\_FIELD\_D  
LCD.h, 153

LCD\_FCT\_SET\_FIELD\_DL  
LCD.h, 154

LCD\_FCT\_SET\_FIELD\_F  
LCD.h, 154

LCD\_FCT\_SET\_FIELD\_N  
LCD.h, 154

LCD\_INST\_CLR\_DISPLAY\_BIT  
LCD.h, 154

LCD\_INST\_DISPLAY\_CTRL  
LCD.h, 154

LCD\_INST\_ENTRY\_MODE\_SET  
LCD.h, 155

LCD\_INST\_FUNCTION\_SET  
LCD.h, 155

LCD\_INST\_SET\_DDRAM\_ADDR  
LCD.h, 155

LCD\_RAM\_1\_LINE\_MAX  
LCD.h, 155

LCD\_RAM\_1\_LINE\_MIN  
LCD.h, 155

LCD\_RAM\_2\_LINES\_MAX\_1  
LCD.h, 156

LCD\_RAM\_2\_LINES\_MAX\_2  
LCD.h, 156

LCD\_RAM\_2\_LINES\_MIN\_1  
LCD.h, 156

LCD\_RAM\_2\_LINES\_MIN\_2  
LCD.h, 156

LCD\_SIZE\_NB\_CHAR\_PER\_LINE  
LCD.h, 156

LCD\_SIZE\_NB\_LINES  
LCD.h, 157

LCD\_WAIT\_CLR\_RETURN  
LCD.h, 157

LCD\_WAIT\_OTHER\_MODES  
LCD.h, 157

LCD\_init\_cnf  
DisplayManagement.h, 142

LCD, 40

- backlight\_enable, 55
- cnfCursorBlink, 55
- cnfCursorOnOff, 55
- cnfDisplayOnOff, 55
- cnfEntryModeDir, 56
- cnfEntryModeShift, 56
- cnfFontType, 56
- cnfI2C\_addr, 56
- cnfLineNumber, 56
- command, 43
- ConfigureBacklight, 43
- ConfigureCursorBlink, 44
- ConfigureCursorOnOff, 45
- ConfigureDisplayOnOff, 45
- ConfigureEntryModeDir, 46
- ConfigureEntryModeShift, 47
- ConfigureFontType, 47
- ConfigureI2CAddr, 48
- ConfigureLineNumber, 49
- ddram\_addr, 57
- GetDDRAMAddress, 49
- GetLineNumberCnf, 50
- InitializeScreen, 50

LCD, 42

- SetDDRAMAddress, 51
- write, 52
- write4bits, 53
- WriteInRam, 54

LED\_PORT

- keepAliveLed.h, 146

LL\_shift\_data\_ptr

- DisplayInterface, 29

LLElementCompare

- DisplayInterface, 27
- scheduler, 67

last\_sum\_value

- CpuLoad, 8

launchPeriodicTasks

- scheduler, 67

line

- T\_Display\_shift\_data, 82

lineEmptyTab

- DisplayInterface, 29

lineNumber\_cnf

- T\_LCD\_conf\_struct, 85

LinkedList, 57  
     AttachNewElement, 59  
     curElement\_ptr, 62  
     firstElement, 63  
     getCurrentElement, 60  
     IsLLEmpty, 60  
     LinkedList, 59  
     MoveToNextElement, 61  
     RemoveElement, 61  
     ResetElementPtr, 62  
     T\_LL\_element, 58  
 LinkedList.cpp, 159  
 LinkedList.h, 159  
     CompareFctPtr\_t, 160  
 LinkedList::T\_LL\_element, 85  
     data\_ptr, 86  
     nextElement, 86  
  
 MAX\_WAIT\_TIME\_US  
     dht22.cpp, 128  
 main  
     main.cpp, 162  
 main.cpp, 160  
     ISR, 161  
     main, 162  
 main.h, 163  
 max\_load  
     CpuLoad, 9  
 MoveToNextElement  
     LinkedList, 61  
  
 NB\_OF\_SAMPLES  
     CpuLoad.h, 124  
 nextElement  
     LinkedList::T\_LL\_element, 86  
  
 operator delete  
     operators.cpp, 164  
     operators.h, 167  
 operator new  
     operators.cpp, 164  
     operators.h, 167  
 operators.cpp, 163  
     operator delete, 164  
     operator new, 164  
 operators.h, 166  
     operator delete, 167  
     operator new, 167  
  
 p\_DisplayInterface  
     T\_ASW\_cnf\_struct, 78  
 p\_DisplayManagement  
     T\_ASW\_cnf\_struct, 79  
 p\_TempSensor  
     T\_ASW\_cnf\_struct, 79  
 p\_cupload  
     T\_BSW\_cnf\_struct, 80  
 p\_dht22  
     T\_BSW\_cnf\_struct, 80  
  
     p\_dio  
         T\_BSW\_cnf\_struct, 81  
     p\_display\_ift  
         DisplayManagement, 34  
     p\_i2c  
         T\_BSW\_cnf\_struct, 81  
     p\_keepAliveLed  
         T\_ASW\_cnf\_struct, 79  
     p\_lcd  
         DisplayInterface, 29  
         T\_BSW\_cnf\_struct, 81  
     p\_scheduler  
         scheduler.cpp, 168  
         scheduler.h, 170  
     p\_tempSensor  
         DisplayManagement, 34  
     p\_timer  
         T\_BSW\_cnf\_struct, 81  
     p\_usart  
         T\_BSW\_cnf\_struct, 81  
     p\_usartDebug  
         T\_ASW\_cnf\_struct, 79  
 PERIOD\_MS\_TASK\_DISPLAY\_CPU\_LOAD  
     debug.h, 126  
 PERIOD\_MS\_TASK\_DISPLAY\_SENSORS  
     debug.h, 126  
 PERIOD\_MS\_TASK\_LED  
     keepAliveLed.h, 146  
 PERIOD\_MS\_TASK\_TEMP\_SENSOR  
     TempSensor.h, 174  
 PINA\_PTR  
     dio\_reg\_atm2560.h, 135  
 PINB\_PTR  
     dio\_reg\_atm2560.h, 135  
 PINC\_PTR  
     dio\_reg\_atm2560.h, 135  
 PIND\_PTR  
     dio\_reg\_atm2560.h, 135  
 PINx\_addr\_mem  
     dio, 21  
 PINx\_idx\_mem  
     dio, 21  
 PIT\_BEFORE\_INVALID  
     TempSensor.cpp, 173  
 PORT\_CNF\_IN  
     dio.h, 132  
 PORT\_CNF\_OUT  
     dio.h, 132  
 PORT\_A  
     dio.h, 131  
 PORT\_B  
     dio.h, 131  
 PORT\_C  
     dio.h, 131  
 PORT\_D  
     dio.h, 132  
 PORTA\_PTR  
     dio\_reg\_atm2560.h, 135

PORTB\_CNF\_DDRB  
    dio\_port\_cnf.h, 133  
PORTB\_CNF\_PORTB  
    dio\_port\_cnf.h, 133  
PORTB\_PTR  
    dio\_reg\_atm2560.h, 136  
PORTC\_PTR  
    dio\_reg\_atm2560.h, 136  
PORTD\_PTR  
    dio\_reg\_atm2560.h, 136  
PRESCALER\_PERIODIC\_TIMER  
    scheduler.h, 170  
period  
    scheduler::Task\_t, 87  
pit\_number  
    scheduler, 70  
ports\_init  
    dio, 21  
prescaler  
    timer, 100  
  
RS\_PIN  
    LCD.h, 157  
RW\_PIN  
    LCD.h, 157  
read  
    dht22, 11  
read\_humidity  
    TempSensor, 95  
read\_temperature  
    TempSensor, 96  
readTempSensor\_task  
    TempSensor, 93  
RemoveElement  
    LinkedList, 61  
removePeriodicTask  
    scheduler, 68  
ResetElementPtr  
    LinkedList, 62  
  
SLA\_ACK  
    I2C.h, 144  
START  
    I2C.h, 144  
SW\_PERIOD\_MS  
    scheduler.h, 170  
sample\_cnt  
    CpuLoad, 9  
sample\_idx  
    CpuLoad, 9  
sample\_mem  
    CpuLoad, 9  
scheduler, 63  
    addPeriodicTask, 65  
    getPitNumber, 66  
    LLElementCompare, 67  
    launchPeriodicTasks, 67  
    pit\_number, 70  
    removePeriodicTask, 68  
  
scheduler, 65  
startScheduling, 69  
Task\_t, 64  
TasksLL\_ptr, 70  
scheduler.cpp, 167  
    p\_scheduler, 168  
scheduler.h, 169  
    p\_scheduler, 170  
PRESCALER\_PERIODIC\_TIMER, 170  
SW\_PERIOD\_MS, 170  
TIMER\_CTC\_VALUE, 170  
TaskPtr\_t, 170  
scheduler::Task\_t, 86  
    period, 87  
    TaskPtr, 87  
sendBool  
    UsartDebug, 111  
sendInteger  
    UsartDebug, 111  
sendString  
    UsartDebug, 112  
setBaudRate  
    uart, 102  
setBitRate  
    I2C, 36  
SetDDRAMAddress  
    LCD, 51  
setTxAddress  
    I2C, 37  
setValidity  
    TempSensor, 94  
shiftLine\_task  
    DisplayInterface, 28  
size  
    String, 77  
    T\_Display\_shift\_data, 82  
startScheduling  
    scheduler, 69  
startTimer1  
    timer, 99  
stopTimer1  
    timer, 100  
str\_cur\_ptr  
    T\_Display\_shift\_data, 83  
str\_debug\_main\_menu  
    debug.cpp, 125  
str\_start\_ptr  
    T\_Display\_shift\_data, 83  
String, 70  
    appendInteger, 72  
    appendString, 73  
    Clear, 74  
    ComputeStringSize, 74  
    getSize, 76  
    getString, 76  
    size, 77  
    String, 71, 72  
    string, 77

string  
 String, 77  
**String.cpp**, 171  
**String.h**, 172

**T\_ASW\_cnf\_struct**, 78  
 p\_DisplayInterface, 78  
 p\_DisplayManagement, 79  
 p\_TempSensor, 79  
 p\_keepAliveLed, 79  
 p\_usartDebug, 79

**T\_BSW\_cnf\_struct**, 80  
 p\_cpupload, 80  
 p\_dht22, 80  
 p\_dio, 81  
 p\_i2c, 81  
 p\_lcd, 81  
 p\_timer, 81  
 p\_usart, 81

**T\_Display\_shift\_data**, 82  
 line, 82  
 size, 82  
 str\_cur\_ptr, 83  
 str\_start\_ptr, 83  
 temporization, 83

**T\_DisplayInterface\_LineDisplayMode**  
 DisplayInterface.h, 139

**T\_LCD\_command**  
 LCD.h, 158

**T\_LCD\_conf\_struct**, 83  
 backlight\_en, 84  
 cursor\_en, 84  
 cursorBlink\_en, 84  
 display\_en, 84  
 entryModeDir, 84  
 entryModeShift, 84  
 fontType\_cnf, 85  
 i2c\_addr, 85  
 lineNumber\_cnf, 85

**T\_LCD\_config\_mode**  
 LCD.h, 158

**T\_LCD\_ram\_area**  
 LCD.h, 158

**T\_LL\_element**  
 LinkedList, 58

**TIMER\_CTC\_VALUE**  
 scheduler.h, 170

**Task\_t**  
 scheduler, 64

**TaskPtr**  
 scheduler::Task\_t, 87

**TaskPtr\_t**  
 scheduler.h, 170

**TasksLL\_ptr**  
 scheduler, 70

**tempDisplayString**  
 DisplayManagement.h, 142

**TempSensor**, 87  
 GetHumDecimal, 89

GetHumInteger, 90  
 getHumPtr, 90  
 getHumidity, 89  
 getTemp, 91  
 GetTempDecimal, 91  
 GetTempInteger, 92  
 getTempPtr, 92  
 GetValidity, 93  
 read\_humidity, 95  
 read\_temperature, 96  
 readTempSensor\_task, 93  
 setValidity, 94  
 TempSensor, 89  
 updateLastValidValues, 95  
 valid\_hum, 96  
 valid坑, 96  
 valid\_temp, 96  
 validity, 96  
 validity\_last\_read, 97

**TempSensor.cpp**, 172  
 PIT\_BEFORE\_INVALID, 173

**TempSensor.h**, 174  
 PERIOD\_MS\_TASK\_TEMP\_SENSOR, 174

temporization  
 T\_Display\_shift\_data, 83

**timer**, 97  
 configureTimer1, 98  
 getTimer1Value, 99  
 prescaler, 100  
 startTimer1, 99  
 stopTimer1, 100  
 timer, 98

**timer.cpp**, 175

**timer.h**, 175

**tx\_address**  
 I2C, 38

**USART\_BAUDRATE**  
 bsw.h, 121

updateLastValidValues  
 TempSensor, 95

**uart**, 101  
 BaudRate, 105  
 setBaudRate, 102  
 usart, 101  
 usart\_init, 102  
 usart\_read, 103  
 usart\_sendString, 103  
 usart\_transmit, 104

**uart.cpp**, 176

**uart.h**, 177

**uart\_init**  
 usart, 102

**uart\_read**  
 usart, 103

**uart\_sendString**  
 usart, 103

**uart\_transmit**  
 usart, 104

UsartDebug, 105  
    activateDebugMode, 107  
    debug\_state, 113  
    debugModeActive\_F, 113  
    DebugModeManagement, 107  
    DisplayCPULoad\_task, 109  
    DisplaySensors\_task, 109  
    isDebugModeActive, 110  
    sendBool, 111  
    sendInteger, 111  
    sendString, 112  
    UsartDebug, 106

valid\_hum  
    TempSensor, 96

valid\_pit  
    TempSensor, 96

valid\_temp  
    TempSensor, 96

validity  
    TempSensor, 96

validity\_last\_read  
    TempSensor, 97

write  
    LCD, 52

write4bits  
    LCD, 53

writeByte  
    I2C, 37

WriteInRam  
    LCD, 54