

Arduino

1.0

Generated by Doxygen 1.8.14

Contents

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

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

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

3.6.3.3	dio_getPort_fast()	37
3.6.3.4	dio_invertPort()	38
3.6.3.5	dio_memorizePINaddress()	38
3.6.3.6	dio_setPort()	39
3.6.3.7	getDDRxAddress()	40
3.6.3.8	getPINxAddress()	41
3.6.3.9	getPORTxAddress()	41
3.6.3.10	ports_init()	42
3.6.4	Member Data Documentation	42
3.6.4.1	PINx_addr_mem	42
3.6.4.2	PINx_idx_mem	43
3.7	DisplayInterface Class Reference	43
3.7.1	Detailed Description	44
3.7.2	Constructor & Destructor Documentation	45
3.7.2.1	DisplayInterface()	45
3.7.3	Member Function Documentation	45
3.7.3.1	ClearFullScreen()	45
3.7.3.2	ClearLine()	46
3.7.3.3	ClearStringInDataStruct()	47
3.7.3.4	DisplayFullLine()	48
3.7.3.5	FindFirstCharAddr()	49
3.7.3.6	getDisplayDataPtr()	49
3.7.3.7	IsLineEmpty()	50
3.7.3.8	RefreshLine()	50
3.7.3.9	setLineAlignment()	51
3.7.3.10	setLineAlignmentAndRefresh()	52
3.7.3.11	shiftLine_task()	52
3.7.3.12	updateLineAndRefresh()	53
3.7.4	Member Data Documentation	54
3.7.4.1	display_data	54

3.7.4.2	dummy	54
3.7.4.3	isShiftInProgress	54
3.7.4.4	p_lcd	55
3.8	DisplayManagement Class Reference	55
3.8.1	Detailed Description	56
3.8.2	Constructor & Destructor Documentation	56
3.8.2.1	DisplayManagement()	56
3.8.3	Member Function Documentation	57
3.8.3.1	DisplaySensorData_Task()	57
3.8.3.2	GetIftPointer()	58
3.8.3.3	GetTempSensorPtr()	59
3.8.3.4	RemoveWelcomeMessage_Task()	59
3.8.4	Member Data Documentation	60
3.8.4.1	p_display_ift	60
3.8.4.2	p_tempSensor	60
3.9	I2C Class Reference	60
3.9.1	Detailed Description	61
3.9.2	Constructor & Destructor Documentation	61
3.9.2.1	I2C()	61
3.9.3	Member Function Documentation	62
3.9.3.1	initializeBus()	62
3.9.3.2	setBitRate()	62
3.9.3.3	setTxAddress()	63
3.9.3.4	writeByte()	63
3.9.4	Member Data Documentation	64
3.9.4.1	bitrate	64
3.9.4.2	tx_address	64
3.10	keepAliveLed Class Reference	64
3.10.1	Detailed Description	65
3.10.2	Constructor & Destructor Documentation	65

3.10.2.1	keepAliveLed()	65
3.10.3	Member Function Documentation	65
3.10.3.1	blinkLed_task()	66
3.11	LCD Class Reference	66
3.11.1	Detailed Description	68
3.11.2	Constructor & Destructor Documentation	68
3.11.2.1	LCD()	68
3.11.3	Member Function Documentation	69
3.11.3.1	command()	69
3.11.3.2	ConfigureBacklight()	70
3.11.3.3	ConfigureCursorBlink()	71
3.11.3.4	ConfigureCursorOnOff()	71
3.11.3.5	ConfigureDisplayOnOff()	72
3.11.3.6	ConfigureEntryModeDir()	72
3.11.3.7	ConfigureEntryModeShift()	73
3.11.3.8	ConfigureFontType()	74
3.11.3.9	ConfigureI2CAddr()	75
3.11.3.10	ConfigureLineNumber()	75
3.11.3.11	GetDDRAMAddress()	76
3.11.3.12	GetLineNumberCnf()	76
3.11.3.13	InitializeScreen()	77
3.11.3.14	SetDDRAMAddress()	77
3.11.3.15	write()	78
3.11.3.16	write4bits()	79
3.11.3.17	WriteInRam()	80
3.11.4	Member Data Documentation	80
3.11.4.1	backlight_enable	81
3.11.4.2	cnfCursorBlink	81
3.11.4.3	cnfCursorOnOff	81
3.11.4.4	cnfDisplayOnOff	81

3.11.4.5	cnfEntryModeDir	81
3.11.4.6	cnfEntryModeShift	82
3.11.4.7	cnfFontType	82
3.11.4.8	cnfI2C_addr	82
3.11.4.9	cnfLineNumber	82
3.11.4.10	ddram_addr	82
3.11.4.11	i2c_drv_ptr	83
3.12	LinkedList Class Reference	83
3.12.1	Detailed Description	84
3.12.2	Member Typedef Documentation	84
3.12.2.1	T_LL_element	85
3.12.3	Constructor & Destructor Documentation	85
3.12.3.1	LinkedList()	85
3.12.3.2	~LinkedList()	85
3.12.4	Member Function Documentation	86
3.12.4.1	AttachNewElement()	86
3.12.4.2	FindElement()	86
3.12.4.3	getCurrentElement()	87
3.12.4.4	IsLLEmpty()	88
3.12.4.5	MoveToNextElement()	88
3.12.4.6	RemoveElement()	89
3.12.4.7	ResetElementPtr()	89
3.12.5	Member Data Documentation	90
3.12.5.1	curElement_ptr	90
3.12.5.2	firstElement	90
3.13	scheduler Class Reference	90
3.13.1	Detailed Description	91
3.13.2	Member Typedef Documentation	92
3.13.2.1	Task_t	92
3.13.3	Constructor & Destructor Documentation	92

3.13.3.1 <code>scheduler()</code>	92
3.13.4 Member Function Documentation	92
3.13.4.1 <code>addPeriodicTask()</code>	92
3.13.4.2 <code>getPitNumber()</code>	93
3.13.4.3 <code>getTaskCount()</code>	94
3.13.4.4 <code>launchPeriodicTasks()</code>	94
3.13.4.5 <code>LLElementCompare()</code>	95
3.13.4.6 <code>removePeriodicTask()</code>	95
3.13.4.7 <code>startScheduling()</code>	96
3.13.4.8 <code>updateTaskPeriod()</code>	97
3.13.5 Member Data Documentation	98
3.13.5.1 <code>pit_number</code>	98
3.13.5.2 <code>task_count</code>	98
3.13.5.3 <code>TasksLL_ptr</code>	98
3.14 String Class Reference	99
3.14.1 Detailed Description	99
3.14.2 Constructor & Destructor Documentation	100
3.14.2.1 <code>String() [1/2]</code>	100
3.14.2.2 <code>String() [2/2]</code>	100
3.14.2.3 <code>~String()</code>	101
3.14.3 Member Function Documentation	101
3.14.3.1 <code>appendBool()</code>	101
3.14.3.2 <code>appendChar()</code>	102
3.14.3.3 <code>appendInteger()</code>	103
3.14.3.4 <code>appendString()</code>	104
3.14.3.5 <code>Clear()</code>	104
3.14.3.6 <code>ComputeStringSize()</code>	105
3.14.3.7 <code>getSize()</code>	106
3.14.3.8 <code>getString()</code>	106
3.14.4 Member Data Documentation	107

3.14.4.1 size	107
3.14.4.2 string	107
3.15 T_ASW_init_cnf Struct Reference	107
3.15.1 Detailed Description	107
3.15.2 Member Data Documentation	108
3.15.2.1 isDebugEnabled	108
3.15.2.2 isDisplayActivated	108
3.15.2.3 isLEDActivated	108
3.15.2.4 isTempSensorActivated	108
3.16 T_display_data Struct Reference	109
3.16.1 Detailed Description	109
3.16.2 Member Data Documentation	109
3.16.2.1 alignment	110
3.16.2.2 display_str	110
3.16.2.3 isEmpty	110
3.16.2.4 mode	110
3.16.2.5 shift_data	110
3.17 T_Display_shift_data Struct Reference	111
3.17.1 Detailed Description	111
3.17.2 Member Data Documentation	111
3.17.2.1 str_cur_ptr	111
3.17.2.2 str_ptr	112
3.17.2.3 temporization	112
3.18 T_LCD_conf_struct Struct Reference	112
3.18.1 Detailed Description	112
3.18.2 Member Data Documentation	113
3.18.2.1 backlight_en	113
3.18.2.2 cursor_en	113
3.18.2.3 cursorBlink_en	113
3.18.2.4 display_en	113

3.18.2.5 entryModeDir	113
3.18.2.6 entryModeShift	114
3.18.2.7 fontType_cnf	114
3.18.2.8 i2c_addr	114
3.18.2.9 i2c_bitrate	114
3.18.2.10 lineNumber_cnf	114
3.19 LinkedList::T_LL_element Struct Reference	115
3.19.1 Detailed Description	115
3.19.2 Member Data Documentation	115
3.19.2.1 data_ptr	115
3.19.2.2 nextElement	115
3.20 scheduler::Task_t Struct Reference	116
3.20.1 Detailed Description	116
3.20.2 Member Data Documentation	116
3.20.2.1 period	116
3.20.2.2 TaskPtr	116
3.21 TempSensor Class Reference	116
3.21.1 Detailed Description	117
3.21.2 Constructor & Destructor Documentation	118
3.21.2.1 TempSensor()	118
3.21.3 Member Function Documentation	118
3.21.3.1 GetHumDecimal()	118
3.21.3.2 getHumidity()	119
3.21.3.3 GetHumInteger()	119
3.21.3.4 getHumPtr()	120
3.21.3.5 getTaskPeriod()	120
3.21.3.6 getTemp()	120
3.21.3.7 GetTempDecimal()	121
3.21.3.8 GetTempInteger()	121
3.21.3.9 getTempPtr()	122

3.21.3.10 <code>GetValidity()</code>	122
3.21.3.11 <code>readTempSensor_task()</code>	123
3.21.3.12 <code>setValidity()</code>	123
3.21.3.13 <code>updateLastValidValues()</code>	124
3.21.3.14 <code>updateTaskPeriod()</code>	124
3.21.4 Member Data Documentation	125
3.21.4.1 <code>read_humidity</code>	125
3.21.4.2 <code>read_temperature</code>	126
3.21.4.3 <code>task_period</code>	126
3.21.4.4 <code>valid_hum</code>	126
3.21.4.5 <code>valid_pit</code>	126
3.21.4.6 <code>valid_temp</code>	126
3.21.4.7 <code>validity</code>	127
3.21.4.8 <code>validity_last_read</code>	127
3.22 timer Class Reference	127
3.22.1 Detailed Description	128
3.22.2 Constructor & Destructor Documentation	128
3.22.2.1 <code>timer()</code>	128
3.22.3 Member Function Documentation	128
3.22.3.1 <code>configureTimer1()</code>	128
3.22.3.2 <code>getTimer1Value()</code>	129
3.22.3.3 <code>startTimer1()</code>	130
3.22.3.4 <code>stopTimer1()</code>	130
3.22.4 Member Data Documentation	130
3.22.4.1 <code>prescaler</code>	130
3.23 usart Class Reference	131
3.23.1 Detailed Description	131
3.23.2 Constructor & Destructor Documentation	131
3.23.2.1 <code>usart()</code>	131
3.23.3 Member Function Documentation	132

3.23.3.1	setBaudRate()	132
3.23.3.2	usart_init()	133
3.23.3.3	usart_read()	133
3.23.3.4	usart_sendByte()	133
3.23.3.5	usart_sendString()	134
3.23.3.6	usart_transmit()	135
3.23.4	Member Data Documentation	136
3.23.4.1	BaudRate	136
3.24	Watchdog Class Reference	136
3.24.1	Detailed Description	137
3.24.2	Constructor & Destructor Documentation	137
3.24.2.1	Watchdog() [1/2]	137
3.24.2.2	Watchdog() [2/2]	138
3.24.3	Member Function Documentation	138
3.24.3.1	disable()	138
3.24.3.2	enable()	139
3.24.3.3	extractResetReason()	140
3.24.3.4	getResetReason()	140
3.24.3.5	getTMOValue()	141
3.24.3.6	isEnabled()	141
3.24.3.7	reset()	142
3.24.3.8	SwitchWdg()	142
3.24.3.9	SystemReset()	143
3.24.3.10	timeoutUpdate()	144
3.24.4	Member Data Documentation	144
3.24.4.1	isActive	144
3.24.4.2	reset_reason	145
3.24.4.3	tmo_value	145

4 File Documentation	147
4.1 asw.cpp File Reference	147
4.1.1 Detailed Description	148
4.1.2 Function Documentation	148
4.1.2.1 asw_init()	148
4.2 asw.h File Reference	149
4.2.1 Detailed Description	149
4.2.2 Function Documentation	149
4.2.2.1 asw_init()	150
4.3 bsw.cpp File Reference	150
4.3.1 Detailed Description	151
4.3.2 Function Documentation	151
4.3.2.1 bsw_init()	151
4.4 bsw.h File Reference	152
4.4.1 Detailed Description	152
4.4.2 Function Documentation	152
4.4.2.1 bsw_init()	153
4.5 CpuLoad.cpp File Reference	153
4.5.1 Detailed Description	154
4.5.2 Variable Documentation	154
4.5.2.1 p_global_BSW_cpupload	154
4.6 CpuLoad.h File Reference	154
4.6.1 Detailed Description	155
4.6.2 Macro Definition Documentation	155
4.6.2.1 NB_OF_SAMPLES	155
4.6.3 Variable Documentation	155
4.6.3.1 p_global_BSW_cpupload	155
4.7 DebugInterface.cpp File Reference	156
4.7.1 Detailed Description	156
4.7.2 Variable Documentation	156

4.7.2.1	p_global_ASW_DebugInterface	156
4.8	DebugInterface.h File Reference	157
4.8.1	Detailed Description	157
4.8.2	Macro Definition Documentation	157
4.8.2.1	USART_BAUDRATE	158
4.8.3	Variable Documentation	158
4.8.3.1	p_global_ASW_DebugInterface	158
4.9	DebugManagement.cpp File Reference	158
4.9.1	Detailed Description	159
4.9.2	Variable Documentation	159
4.9.2.1	p_global_ASW_DebugManagement	159
4.9.2.2	str_debug_info_message_wdg_disabled	160
4.9.2.3	str_debug_info_message_wdg_enabled	160
4.9.2.4	str_debug_info_message_wdg_tmo_updated	160
4.9.2.5	str_debug_info_message_wdg_tmo_value	160
4.9.2.6	str_debug_info_message_wrong_menu_selection	160
4.9.2.7	str_debug_main_menu	161
4.9.2.8	str_debug_wdg_menu	161
4.9.2.9	str_debug_wdg_timeout_update_selection	161
4.10	DebugManagement.h File Reference	162
4.10.1	Detailed Description	162
4.10.2	Macro Definition Documentation	163
4.10.2.1	PERIOD_MS_TASK_DISPLAY_CPU_LOAD	163
4.10.2.2	PERIOD_MS_TASK_DISPLAY_DEBUG_DATA	163
4.10.3	Enumeration Type Documentation	163
4.10.3.1	debug_mgt_main_menu_state_t	163
4.10.3.2	debug_mgt_wdg_state_t	163
4.10.4	Variable Documentation	164
4.10.4.1	p_global_ASW_DebugManagement	164
4.11	dht22.cpp File Reference	164

4.11.1	Detailed Description	165
4.11.2	Macro Definition Documentation	165
4.11.2.1	MAX_WAIT_TIME_US	165
4.11.3	Variable Documentation	165
4.11.3.1	p_global_BSW_dht22	165
4.12	dht22.h File Reference	166
4.12.1	Detailed Description	166
4.12.2	Variable Documentation	166
4.12.2.1	p_global_BSW_dht22	166
4.13	dio.cpp File Reference	167
4.13.1	Detailed Description	167
4.13.2	Variable Documentation	167
4.13.2.1	p_global_BSW_dio	168
4.14	dio.h File Reference	168
4.14.1	Detailed Description	169
4.14.2	Macro Definition Documentation	169
4.14.2.1	DECODE_PIN	169
4.14.2.2	DECODE_PORT	169
4.14.2.3	ENCODE_PORT	169
4.14.2.4	PORT_CNF_IN	170
4.14.2.5	PORT_CNF_OUT	170
4.14.3	Variable Documentation	170
4.14.3.1	p_global_BSW_dio	170
4.15	dio_port_cnf.h File Reference	170
4.15.1	Detailed Description	171
4.15.2	Macro Definition Documentation	171
4.15.2.1	PORT_A	171
4.15.2.2	PORT_B	171
4.15.2.3	PORT_C	172
4.15.2.4	PORT_D	172

4.15.2.5 PORTB_CNF_DDRB	172
4.15.2.6 PORTB_CNF_PORTB	172
4.16 dio_reg_atm2560.h File Reference	173
4.16.1 Detailed Description	173
4.16.2 Macro Definition Documentation	173
4.16.2.1 DDRA_PTR	174
4.16.2.2 DDRB_PTR	174
4.16.2.3 DDRC_PTR	174
4.16.2.4 DDRD_PTR	174
4.16.2.5 PINA_PTR	174
4.16.2.6 PINB_PTR	175
4.16.2.7 PINC_PTR	175
4.16.2.8 PIND_PTR	175
4.16.2.9 PORTA_PTR	175
4.16.2.10 PORTB_PTR	175
4.16.2.11 PORTC_PTR	176
4.16.2.12 PORTD_PTR	176
4.17 DisplayInterface.cpp File Reference	176
4.17.1 Detailed Description	176
4.17.2 Variable Documentation	177
4.17.2.1 p_global_ASW_DisplayInterface	177
4.18 DisplayInterface.h File Reference	177
4.18.1 Detailed Description	178
4.18.2 Macro Definition Documentation	178
4.18.2.1 DISPLAY_LINE_SHIFT_PERIOD_MS	178
4.18.2.2 DISPLAY_LINE_SHIFT_TEMPO_TIME	178
4.18.3 Enumeration Type Documentation	178
4.18.3.1 T_DisplayInterface_LineAlignment	178
4.18.3.2 T_DisplayInterface_LineDisplayMode	179
4.18.4 Variable Documentation	179

4.18.4.1 p_global_ASW_DisplayInterface	179
4.19 DisplayManagement.cpp File Reference	180
4.19.1 Detailed Description	180
4.19.2 Variable Documentation	180
4.19.2.1 p_global_ASW_DisplayManagement	180
4.20 DisplayManagement.h File Reference	181
4.20.1 Detailed Description	181
4.20.2 Macro Definition Documentation	182
4.20.2.1 DISPLAY_MGT_I2C_BITRATE	182
4.20.2.2 DISPLAY_MGT_LCD_I2C_ADDR	182
4.20.2.3 DISPLAY_MGT_LINE_HUM	182
4.20.2.4 DISPLAY_MGT_LINE_TEMP	182
4.20.2.5 DISPLAY_MGT_PERIOD_TASK_SENSOR	182
4.20.2.6 DISPLAY_MGT_PERIOD_WELCOME_MSG_REMOVAL	183
4.20.3 Variable Documentation	183
4.20.3.1 humidityDisplayString	183
4.20.3.2 LCD_init_cnf	183
4.20.3.3 noSensorDisplayString	184
4.20.3.4 p_global_ASW_DisplayManagement	184
4.20.3.5 tempDisplayString	184
4.20.3.6 welcomeMessageString	184
4.21 I2C.cpp File Reference	184
4.21.1 Detailed Description	185
4.21.2 Variable Documentation	185
4.21.2.1 p_global_BSW_i2c	185
4.22 I2C.h File Reference	185
4.22.1 Detailed Description	186
4.22.2 Macro Definition Documentation	186
4.22.2.1 DATA_ACK	186
4.22.2.2 SLA_ACK	186

4.22.2.3 START	187
4.22.3 Variable Documentation	187
4.22.3.1 p_global_BSW_i2c	187
4.23 int.cpp File Reference	187
4.23.1 Detailed Description	188
4.23.2 Function Documentation	188
4.23.2.1 ISR() [1/2]	188
4.23.2.2 ISR() [2/2]	189
4.24 keepAliveLed.cpp File Reference	189
4.24.1 Detailed Description	190
4.24.2 Variable Documentation	190
4.24.2.1 p_global_ASW_keepAliveLed	190
4.25 keepAliveLed.h File Reference	190
4.25.1 Detailed Description	191
4.25.2 Macro Definition Documentation	191
4.25.2.1 LED_PORT	191
4.25.2.2 PERIOD_MS_TASK_LED	191
4.25.3 Variable Documentation	192
4.25.3.1 p_global_ASW_keepAliveLed	192
4.26 LCD.cpp File Reference	192
4.26.1 Detailed Description	192
4.26.2 Variable Documentation	193
4.26.2.1 p_global_BSW_lcd	193
4.27 LCD.h File Reference	193
4.27.1 Detailed Description	195
4.27.2 Macro Definition Documentation	195
4.27.2.1 BACKLIGHT_PIN	195
4.27.2.2 EN_PIN	195
4.27.2.3 LCD_CNF_BACKLIGHT_OFF	195
4.27.2.4 LCD_CNF_BACKLIGHT_ON	196

4.27.2.5 LCD_CNF_CURSOR_BLINK_OFF	196
4.27.2.6 LCD_CNF_CURSOR_BLINK_ON	196
4.27.2.7 LCD_CNF_CURSOR_OFF	196
4.27.2.8 LCD_CNF_CURSOR_ON	196
4.27.2.9 LCD_CNF_DISPLAY_OFF	197
4.27.2.10 LCD_CNF_DISPLAY_ON	197
4.27.2.11 LCD_CNF_ENTRY_MODE_DIRECTION_LEFT	197
4.27.2.12 LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT	197
4.27.2.13 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF	197
4.27.2.14 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON	198
4.27.2.15 LCD_CNF_FONT_5_11	198
4.27.2.16 LCD_CNF_FONT_5_8	198
4.27.2.17 LCD_CNF_ONE_LINE	198
4.27.2.18 LCD_CNF_SHIFT_ID	198
4.27.2.19 LCD_CNF_SHIFT_SH	199
4.27.2.20 LCD_CNF_TWO_LINE	199
4.27.2.21 LCD_DISPLAY_CTRL_FIELD_B	199
4.27.2.22 LCD_DISPLAY_CTRL_FIELD_C	199
4.27.2.23 LCD_DISPLAY_CTRL_FIELD_D	199
4.27.2.24 LCD_FCT_SET_FIELD_DL	200
4.27.2.25 LCD_FCT_SET_FIELD_F	200
4.27.2.26 LCD_FCT_SET_FIELD_N	200
4.27.2.27 LCD_INST_CLR_DISPLAY_BIT	200
4.27.2.28 LCD_INST_DISPLAY_CTRL	200
4.27.2.29 LCD_INST_ENTRY_MODE_SET	201
4.27.2.30 LCD_INST_FUNCTION_SET	201
4.27.2.31 LCD_INST_SET_DDRAM_ADDR	201
4.27.2.32 LCD_RAM_1_LINE_MAX	201
4.27.2.33 LCD_RAM_1_LINE_MIN	201
4.27.2.34 LCD_RAM_2_LINES_MAX_1	202

4.27.2.35 LCD_RAM_2_LINES_MAX_2	202
4.27.2.36 LCD_RAM_2_LINES_MIN_1	202
4.27.2.37 LCD_RAM_2_LINES_MIN_2	202
4.27.2.38 LCD_SIZE_NB_CHAR_PER_LINE	202
4.27.2.39 LCD_SIZE_NB_LINES	203
4.27.2.40 LCD_WAIT_CLR_RETURN	203
4.27.2.41 LCD_WAIT_OTHER_MODES	203
4.27.2.42 RS_PIN	203
4.27.2.43 RW_PIN	203
4.27.3 Enumeration Type Documentation	203
4.27.3.1 T_LCD_command	203
4.27.3.2 T_LCD_config_mode	204
4.27.3.3 T_LCD_ram_area	204
4.27.4 Variable Documentation	204
4.27.4.1 p_global_BSW_lcd	205
4.28 LinkedList.cpp File Reference	205
4.28.1 Detailed Description	205
4.29 LinkedList.h File Reference	206
4.29.1 Detailed Description	206
4.29.2 Typedef Documentation	206
4.29.2.1 CompareFctPtr_t	206
4.30 main.cpp File Reference	207
4.30.1 Detailed Description	207
4.30.2 Macro Definition Documentation	208
4.30.2.1 DEBUG_ACTIVE_PORT	208
4.30.3 Function Documentation	208
4.30.3.1 main()	208
4.30.4 Variable Documentation	208
4.30.4.1 ASW_init_cnf	209
4.30.4.2 isDebugModeActivated	209

4.31 main.h File Reference	209
4.31.1 Detailed Description	210
4.31.2 Variable Documentation	210
4.31.2.1 ASW_init_cnf	210
4.31.2.2 isDebugModeActivated	210
4.32 operators.cpp File Reference	210
4.32.1 Detailed Description	211
4.32.2 Function Documentation	211
4.32.2.1 operator delete()	211
4.32.2.2 operator new()	211
4.33 operators.h File Reference	213
4.33.1 Detailed Description	213
4.33.2 Function Documentation	214
4.33.2.1 operator delete()	214
4.33.2.2 operator new()	214
4.34 scheduler.cpp File Reference	214
4.34.1 Detailed Description	215
4.34.2 Variable Documentation	215
4.34.2.1 p_global_scheduler	215
4.35 scheduler.h File Reference	216
4.35.1 Detailed Description	216
4.35.2 Macro Definition Documentation	217
4.35.2.1 PRESCALER_PERIODIC_TIMER	217
4.35.2.2 SW_PERIOD_MS	217
4.35.2.3 TIMER_CTC_VALUE	217
4.35.3 Typedef Documentation	217
4.35.3.1 TaskPtr_t	217
4.35.4 Variable Documentation	217
4.35.4.1 p_global_scheduler	218
4.36 String.cpp File Reference	218

4.36.1 Detailed Description	218
4.37 String.h File Reference	218
4.37.1 Detailed Description	219
4.38 TempSensor.cpp File Reference	219
4.38.1 Detailed Description	220
4.38.2 Macro Definition Documentation	220
4.38.2.1 PIT_BEFORE_INVALID	220
4.38.3 Variable Documentation	220
4.38.3.1 p_global_ASW_TempSensor	220
4.39 TempSensor.h File Reference	220
4.39.1 Detailed Description	221
4.39.2 Macro Definition Documentation	221
4.39.2.1 DHT22_PORT	221
4.39.2.2 PERIOD_MS_TASK_TEMP_SENSOR	221
4.39.3 Variable Documentation	222
4.39.3.1 p_global_ASW_TempSensor	222
4.40 timer.cpp File Reference	222
4.40.1 Detailed Description	222
4.40.2 Variable Documentation	223
4.40.2.1 p_global_BSW_timer	223
4.41 timer.h File Reference	223
4.41.1 Detailed Description	223
4.41.2 Variable Documentation	224
4.41.2.1 p_global_BSW_timer	224
4.42 usart.cpp File Reference	224
4.42.1 Detailed Description	224
4.42.2 Variable Documentation	225
4.42.2.1 p_global_BSW_usart	225
4.43 usart.h File Reference	225
4.43.1 Detailed Description	225

4.43.2 Variable Documentation	226
4.43.2.1 p_global_BSW_usart	226
4.44 Watchdog.cpp File Reference	226
4.44.1 Detailed Description	227
4.44.2 Macro Definition Documentation	227
4.44.2.1 WDG_TIMEOUT_DEFAULT_MS	227
4.44.3 Function Documentation	227
4.44.3.1 __attribute__()	227
4.44.3.2 get_mcusr()	228
4.44.4 Variable Documentation	228
4.44.4.1 p_global_BSW_wdg	228
4.45 Watchdog.h File Reference	228
4.45.1 Detailed Description	229
4.45.2 Macro Definition Documentation	229
4.45.2.1 WDG_TMO_120MS	230
4.45.2.2 WDG_TMO_15MS	230
4.45.2.3 WDG_TMO_1S	230
4.45.2.4 WDG_TMO_250MS	230
4.45.2.5 WDG_TMO_2S	230
4.45.2.6 WDG_TMO_30MS	231
4.45.2.7 WDG_TMO_4S	231
4.45.2.8 WDG_TMO_500MS	231
4.45.2.9 WDG_TMO_60MS	231
4.45.2.10 WDG_TMO_8S	231
4.45.3 Enumeration Type Documentation	231
4.45.3.1 T_WDG_Reset_type	231
4.45.4 Function Documentation	232
4.45.4.1 __attribute__()	232
4.45.5 Variable Documentation	232
4.45.5.1 p_global_BSW_wdg	232
Index	233

Chapter 1

Class Index

1.1 Class List

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

CpuLoad	Class defining CPU load libraries	5	
	debug_mgt_state_struct_t	Structure containing all debug states	10
DebugInterface	Class used for debugging on usart link	11	
DebugManagement	Debug management class	18	
dht22	DHT 22 driver class	30	
dio	DIO class	34	
DisplayInterface	Display interface services class	43	
DisplayManagement	Display management class	55	
I2C	Two-wire serial interface (I2C) class definition	60	
keepAliveLed	Class for keep-alive LED blinking	64	
LCD	Class for LCD S2004A display driver	66	
LinkedList	Linked list class	83	
scheduler	Scheduler class	90	
String	String management class	99	
T_ASW_init_cnf	ASW initialization configuration structure	107	
T_display_data	Structure containing display data	109	
T_Display_shift_data	Structure containing shift data	111	
T_LCD_conf_struct	Structure defining LCD configuration	112	

LinkedList::T_LL_element	
Type defining a linked list element	115
scheduler::Task_t	
Type defining a task structure	116
TempSensor	
Class for temperature sensor	116
timer	
Class defining a timer	127
usart	
USART serial bus class	131
Watchdog	
Watchdog management class	136

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

asw.cpp	ASW main file	147
asw.h	ASW main header file	149
bsw.cpp	BSW main file	150
bsw.h	BSW main header file	152
CpuLoad.cpp	Defines functions of class CpuLoad	153
CpuLoad.h	CpuLoad class header file	154
DebugInterface.cpp	This file defines classes for log and debug data transmission on USART link	156
DebugInterface.h	Header file for debug and logging functions	157
DebugManagement.cpp	Debug management class source file	158
DebugManagement.h	Debug management class header file	162
dht22.cpp	This file defines classes for DHT22 driver	164
dht22.h	DHT22 driver header file	166
dio.cpp	DIO library	167
dio.h	DIO library header file	168
dio_port_cnf.h	Digital ports configuration file	170
dio_reg_atm2560.h	Defines DIO register addresses for ATMEGA2560	173
DisplayInterface.cpp	Source code file for display services	176
DisplayInterface.h	DisplayInterface class header file	177

DisplayManagement.cpp	Display management source file	180
DisplayManagement.h	Display management class header file	181
I2C.cpp	Two-wire interface (I2C) source file	184
I2C.h	I2C class header file	185
int.cpp	Interrupt management source file	187
keepAliveLed.cpp	Definition of function for class keepAliveLed	189
keepAliveLed.h	Class keepAliveLed header file	190
LCD.cpp	LCD class source file	192
LCD.h	LCD class header file	193
LinkedList.cpp	Linked List library source file	205
LinkedList.h	Linked List library header file	206
main.cpp	Background task file	207
main.h	Background task header file	209
operators.cpp	C++ operators definitions	210
operators.h	C++ operators definitions header file	213
scheduler.cpp	Defines scheduler class	214
scheduler.h	Scheduler class header file	216
String.cpp	String class source file	218
String.h	String class header file	218
TempSensor.cpp	Defines function of class TempSensor	219
TempSensor.h	Class TempSensor header file	220
timer.cpp	Defines function for class timer	222
timer.h	Timer class header file	223
uart.cpp	BSW library for USART	224
uart.h	Header file for USART library	225
Watchdog.cpp	Class Watchdog source code file	226
Watchdog.h	Class Watchdog header file	228

Chapter 3

Class Documentation

3.1 CpuLoad Class Reference

Class defining CPU load libraries.

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

Public Member Functions

- [CpuLoad \(\)](#)
CpuLoad class constructor.
- [void ComputeCPULoad \(\)](#)
Computes current CPU load.
- [uint8_t getCurrentCPULoad \(\)](#)
Get current CPU load value.
- [uint8_t getAverageCPULoad \(\)](#)
Get average CPU load value.
- [uint8_t getMaxCPULoad \(\)](#)
Get maximum CPU load value.

Private Attributes

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

3.1.1 Detailed Description

Class defining CPU load libraries.

This class defines tools to compute and monitor CPU load.

Definition at line 19 of file CpuLoad.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 CpuLoad()

```
CpuLoad::CpuLoad ( )
```

[CpuLoad](#) class constructor.

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

Returns

Nothing

Definition at line 20 of file CpuLoad.cpp.

3.1.3 Member Function Documentation

3.1.3.1 ComputeCPUload()

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

Computes current CPU load.

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

Returns

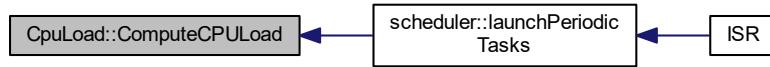
Nothing

Definition at line 40 of file CpuLoad.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.3.2 getAverageCPULoad()

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

Get average CPU load value.

This function returns the average CPU load value

Returns

Average CPU load value

Definition at line 58 of file CpuLoad.h.

Here is the caller graph for this function:



3.1.3.3 getCurrentCPULoad()

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

Get current CPU load value.

This function returns the current CPU load value

Returns

Current CPU load value

Definition at line 47 of file CpuLoad.h.

Here is the caller graph for this function:



3.1.3.4 getMaxCPUload()

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

Get maximum CPU load value.

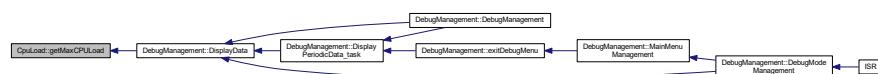
This function returns the maximum CPU load value

Returns

Maximum CPU load value

Definition at line 69 of file CpuLoad.h.

Here is the caller graph for this function:



3.1.4 Member Data Documentation

3.1.4.1 avg_load

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

Average CPU load based on the last 50 cycles

Definition at line 76 of file CpuLoad.h.

3.1.4.2 current_load

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

Current CPU load (load of last cycle)

Definition at line 75 of file CpuLoad.h.

3.1.4.3 last_sum_value

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

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

Definition at line 81 of file CpuLoad.h.

3.1.4.4 max_load

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

Maximum CPU load since power on

Definition at line 77 of file CpuLoad.h.

3.1.4.5 sample_cnt

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

Number of samples used to compute average load

Definition at line 78 of file CpuLoad.h.

3.1.4.6 sample_idx

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

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

Definition at line 80 of file CpuLoad.h.

3.1.4.7 sample_mem

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

Memorization of the last NB_OF_SAMPLES measures

Definition at line 79 of file CpuLoad.h.

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

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

3.2 debug_mgt_state_struct_t Struct Reference

Structure containing all debug states.

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

Public Attributes

- `debug_mgt_main_menu_state_t main_state`
- `debug_mgt_wdg_state_t wdg_state`

3.2.1 Detailed Description

Structure containing all debug states.

Definition at line 40 of file DebugManagement.h.

3.2.2 Member Data Documentation

3.2.2.1 main_state

```
debug_mgt_main_menu_state_t debug_mgt_state_struct_t::main_state
```

Current main menu state

Definition at line 42 of file DebugManagement.h.

3.2.2.2 wdg_state

```
debug_mgt_wdg_state_t debug_mgt_state_struct_t::wdg_state
```

Current state of watchdog management

Definition at line 43 of file DebugManagement.h.

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

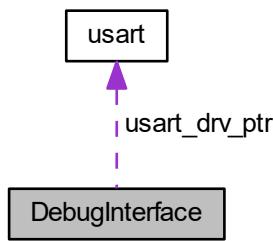
- [DebugManagement.h](#)

3.3 DebugInterface Class Reference

Class used for debugging on usart link.

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

Collaboration diagram for DebugInterface:



Public Member Functions

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

Private Attributes

- [uart * usart_drv_ptr](#)

3.3.1 Detailed Description

Class used for debugging on usart link.

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

Definition at line 21 of file DebugInterface.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 DebugInterface()

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

Class [DebugInterface](#) constructor.

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

Returns

Nothing

Definition at line 22 of file DebugInterface.cpp.

3.3.3 Member Function Documentation

3.3.3.1 ClearScreen()

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

Screen clearing function.

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

Returns

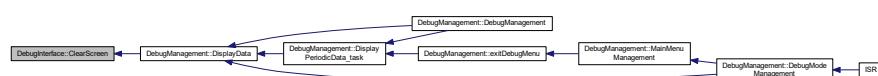
Nothing

Definition at line 77 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.3.3.2 nextLine()

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

Go to next line function.

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

Returns

Nothing

Definition at line 71 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.3.3.3 read()

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

USART read function.

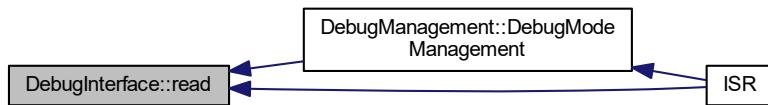
This function will read the last received byte on USART link

Returns

Received byte

Definition at line 82 of file DebugInterface.h.

Here is the caller graph for this function:



3.3.3.4 sendBool()

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

Send a boolean data on USART link.

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

Parameters

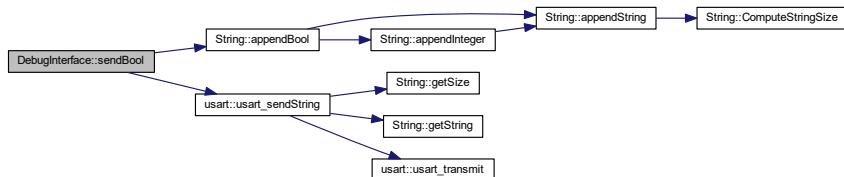
in	<code>data</code>	boolean data to be sent
in	<code>isText</code>	String conversion configuration

Returns

Nothing

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

Here is the call graph for this function:



3.3.3.5 sendChar()

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

Send a single character on USART link.

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

Parameters

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

Returns

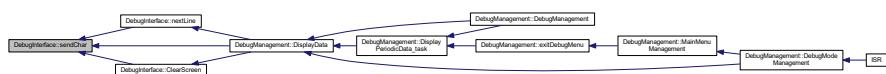
Nothing

Definition at line 44 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.3.3.6 sendInteger()**

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

Send a integer data on USART link.

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

Parameters

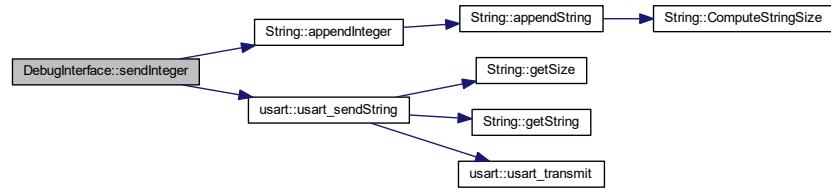
in	data	integer data to be sent
in	base	numerical base used to convert integer into string (between 2 and 36)

Returns

Nothing

Definition at line 49 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.3.3.7 sendString() [1/2]

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

Send a string on USART link.

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

Parameters

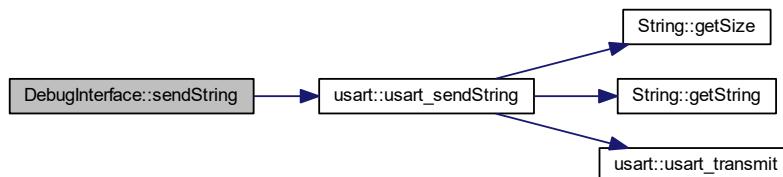
in	<i>str</i>	Pointer to the string being sent
----	------------	----------------------------------

Returns

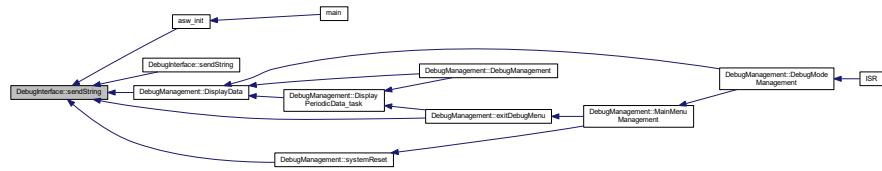
Nothing

Definition at line 31 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.3.3.8 sendString() [2/2]

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

Send a chain of characters on USART link.

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

Parameters

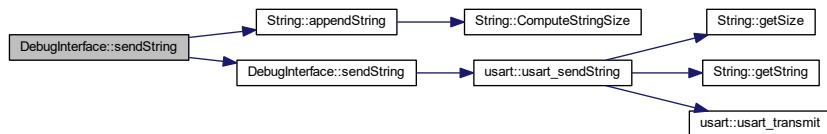
in	<i>str</i>	Pointer to the chain to send.
----	------------	-------------------------------

Returns

Nothing

Definition at line 37 of file DebugInterface.cpp.

Here is the call graph for this function:



3.3.4 Member Data Documentation

3.3.4.1 usart_drv_ptr

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

Pointer to USART driver object

Definition at line 107 of file DebugInterface.h.

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

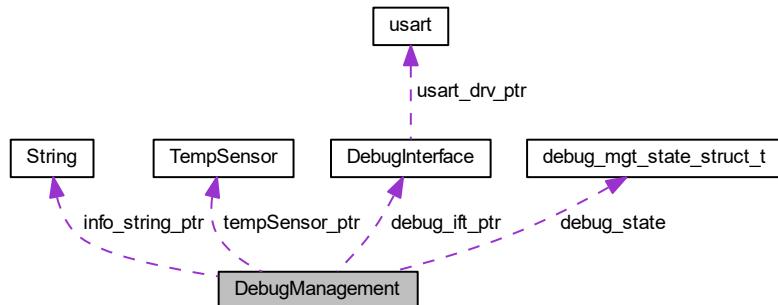
- [DebugInterface.h](#)
- [DebugInterface.cpp](#)

3.4 DebugManagement Class Reference

Debug management class.

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

Collaboration diagram for DebugManagement:



Public Member Functions

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

Static Public Member Functions

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

Displays periodic data on usart link.

Private Member Functions

- void [exitDebugMenu \(\)](#)
Debug menu exit function.
- void [systemReset \(\)](#)
System reset function.
- void [WatchdogMenuManagement \(uint8_t rcv_char\)](#)
Watchdog menu management function.
- bool [MainMenuManagement \(uint8_t rcv_char\)](#)
Main menu management.

Private Attributes

- [DebugInterface * debug_ift_ptr](#)
- [TempSensor * tempSensor_ptr](#)
- [uint8_t * menu_string_ptr](#)
- [String * info_string_ptr](#)
- [debug_mgt_state_struct_t debug_state](#)
- bool [isInfoStringDisplayed](#)

3.4.1 Detailed Description

Debug management class.

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

Definition at line 51 of file `DebugManagement.h`.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 DebugManagement()

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

Class constructor.

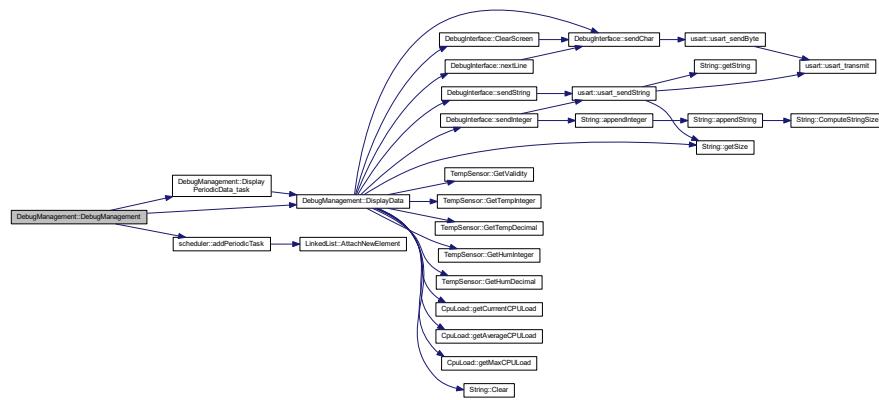
This function initializes the class. If needed, it creates a new instance of debug interface object.

Returns

Nothing

Definition at line 100 of file DebugManagement.cpp.

Here is the call graph for this function:



3.4.3 Member Function Documentation

3.4.3.1 DebugModeManagement()

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

Management of debug mode.

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

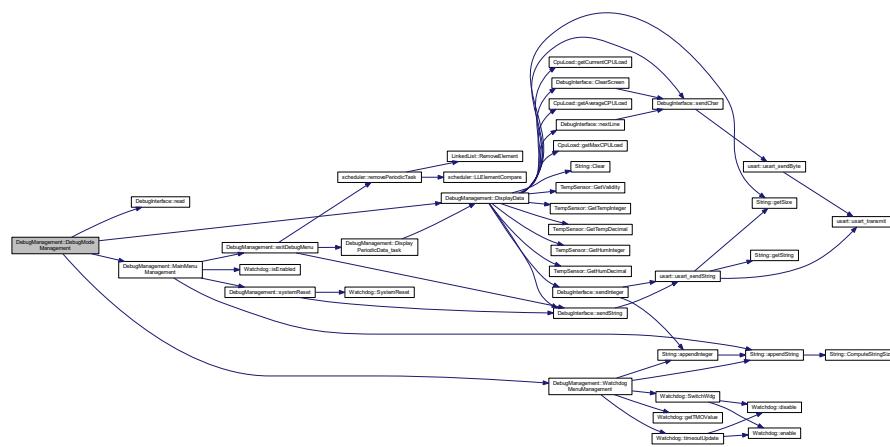
- MAIN_MENU state : handles user choice in main menu and selects next state
- WDG_MENU state : handles user choice in watchdog menu and selects next state
It is called each time a data is received on USART and debug mode is active.

Returns

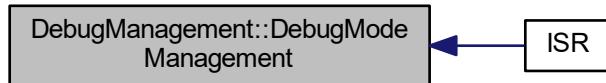
True if the debug mode shall be closed, false otherwise

Definition at line 222 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.2 DisplayData()

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

Displays data on usart link.

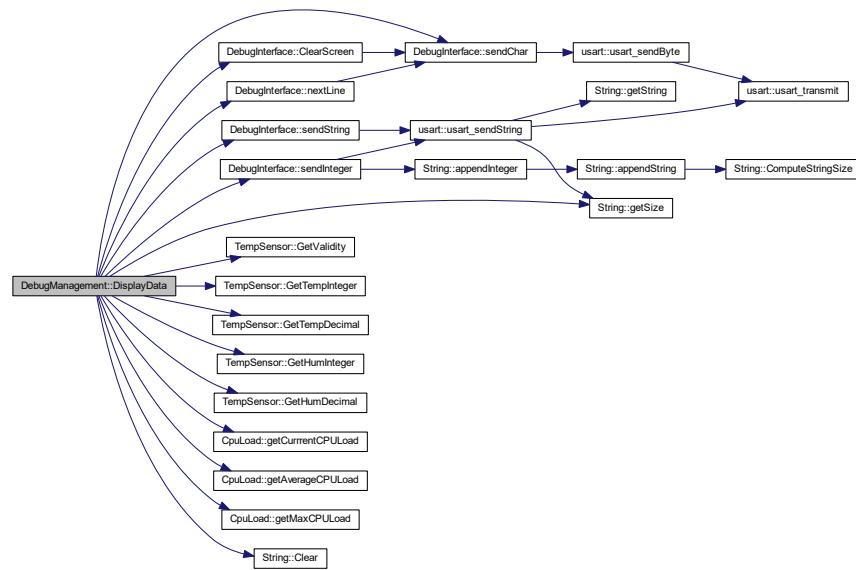
This task displays the menu and periodic data (temperature, humidity and CPU load) on usart screen.

Returns

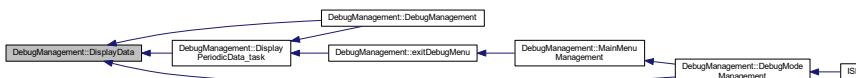
Nothing

Definition at line 129 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.3 DisplayPeriodicData_task()

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

Displays periodic data on uart link.

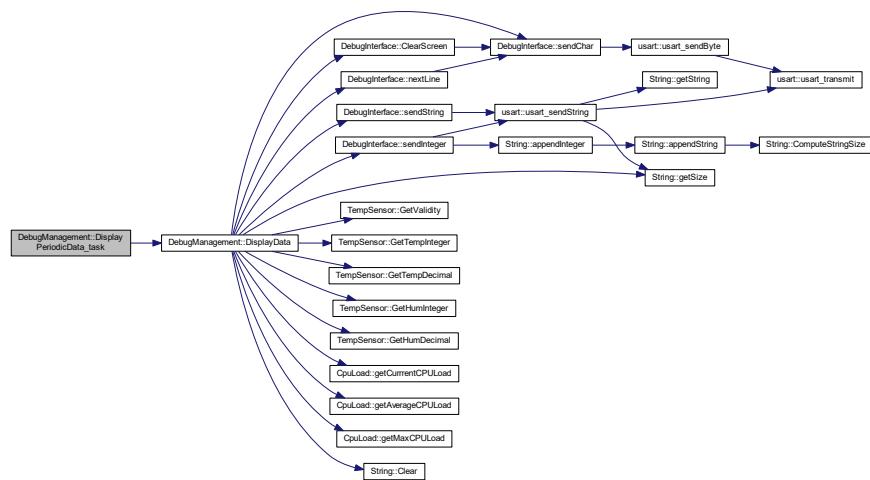
This task displays the menu and periodic data (temperature, humidity and CPU load) on uart screen. It only calls the function `DisplayData`.

Returns

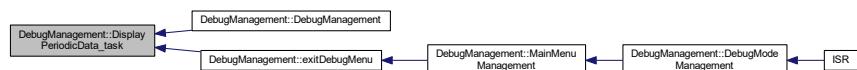
Nothing

Definition at line 214 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.4 exitDebugMenu()

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

Debug menu exit function.

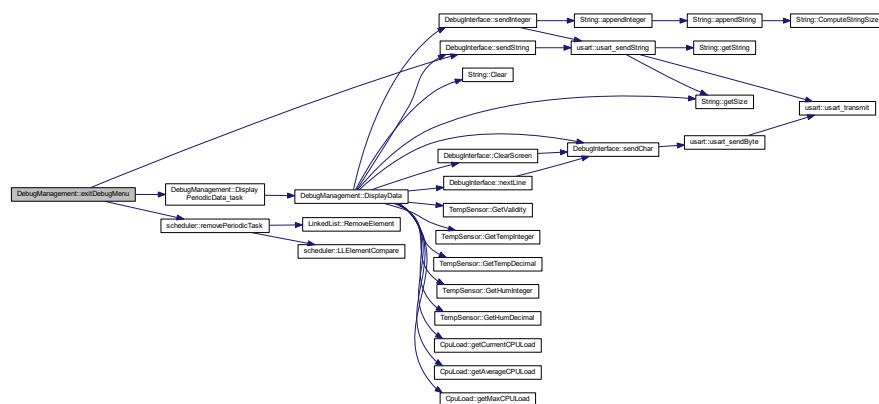
This function prepares the exit of debug menu. It writes the message "Bye !" on the screen and removes the periodic task from the scheduler.

Returns

Nothing.

Definition at line 249 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.4.3.5 getIfpt()**

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

Interface pointer get function.

This function returns the pointer to the debug interface object

Returns

Pointer to debug interface

Definition at line 95 of file DebugManagement.h.

3.4.3.6 getInfoStringPtr()

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

Info string get function.

This function returns the pointer to the info string to display

Returns

Info string pointer

Definition at line 115 of file DebugManagement.h.

3.4.3.7 getMenuStringPtr()

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

Menu string get function.

This function returns the pointer to the menu string to display

Returns

Menu string pointer

Definition at line 105 of file DebugManagement.h.

3.4.3.8 MainMenuManagement()

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

Main menu management.

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

Parameters

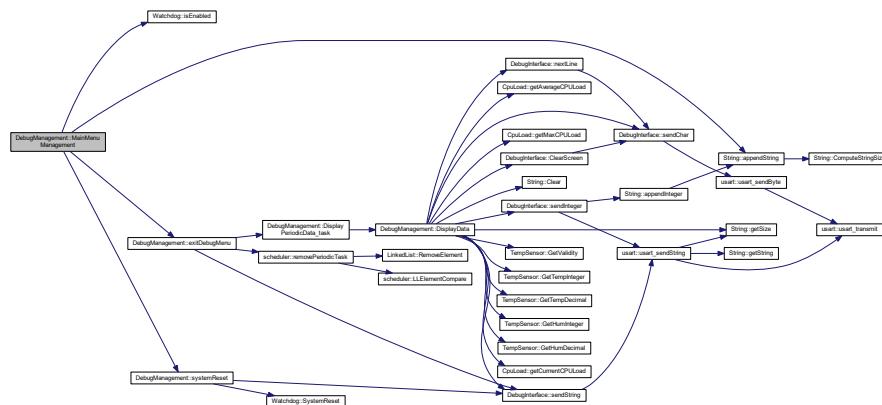
in	<i>rcv_char</i>	Character received on USART bus.
----	-----------------	----------------------------------

Returns

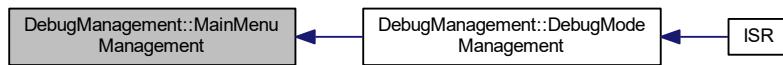
True if the debug mode shall be exited, false otherwise.

Definition at line 366 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.9 setInfoStringPtr()

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

Info message setting function.

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

Parameters

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

Returns

Nothing

Definition at line 126 of file DebugManagement.h.

3.4.3.10 systemReset()

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

System reset function.

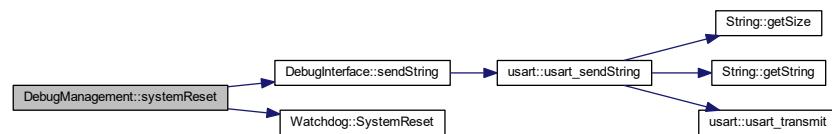
This function provokes a reset of the system. It displays a message on the screen and calls the reset function from watchdog class.

Returns

Nothing.

Definition at line 255 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.11 WatchdogMenuManagement()

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

[Watchdog](#) menu management function.

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

Parameters

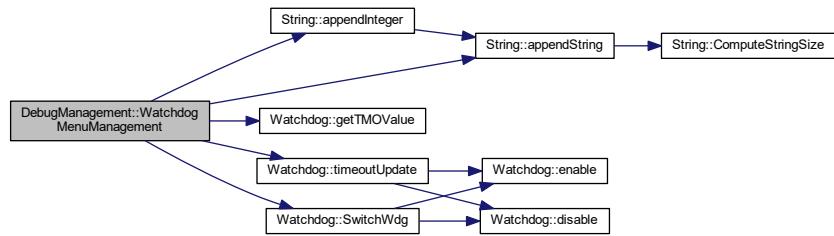
in	<code>rcv_char</code>	Character received on USART bus.
----	-----------------------	----------------------------------

Returns

Nothing.

Definition at line 261 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.4 Member Data Documentation

3.4.4.1 debug_ift_ptr

`DebugInterface* DebugManagement::debug_ift_ptr [private]`

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

Definition at line 133 of file DebugManagement.h.

3.4.4.2 debug_state

`debug_mgt_state_struct_t DebugManagement::debug_state [private]`

Structure containing debug states for each menu

Definition at line 137 of file DebugManagement.h.

3.4.4.3 info_string_ptr

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

Pointer to the info message to display

Definition at line 136 of file DebugManagement.h.

3.4.4.4 isInfoStringDisplayed

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

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

Definition at line 138 of file DebugManagement.h.

3.4.4.5 menu_string_ptr

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

Pointer to the current menu string to display

Definition at line 135 of file DebugManagement.h.

3.4.4.6 tempSensor_ptr

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

Pointer to the temperature sensor object

Definition at line 134 of file DebugManagement.h.

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

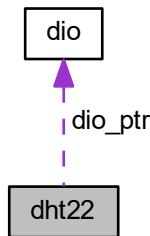
- [DebugManagement.h](#)
- [DebugManagement.cpp](#)

3.5 dht22 Class Reference

DHT 22 driver class.

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

Collaboration diagram for dht22:



Public Member Functions

- [dht22](#) (uint8_t port)
dht22 class constructor.
- bool [read](#) (uint16_t *raw_humidity, uint16_t *raw_temperature)
Reads the data from DHT22.

Private Member Functions

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

Private Attributes

- uint8_t [dht22_port](#)
- [dio](#) * [dio_ptr](#)

3.5.1 Detailed Description

DHT 22 driver class.

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

Definition at line 19 of file dht22.h.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 dht22()

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

[dht22](#) class constructor.

Initializes the class [dht22](#).

Parameters

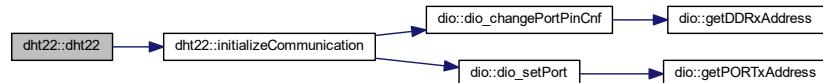
in	<i>port</i>	Encoded configuration of the port used for 1-wire communication.
----	-------------	--

Returns

Nothing

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

Here is the call graph for this function:



3.5.3 Member Function Documentation

3.5.3.1 initializeCommunication()

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

Initializes the communication.

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

Returns

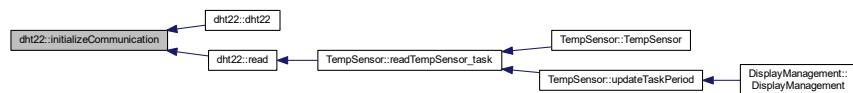
Nothing

Definition at line 201 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.5.3.2 `read()`**

```

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

Reads the data from DHT22.

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

Parameters

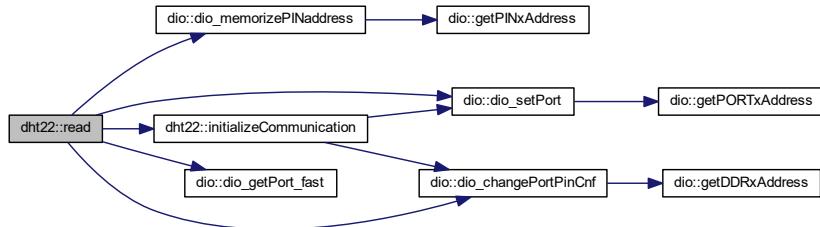
<code>out</code>	<code>raw_humidity</code>	Raw humidity value received from sensor
<code>out</code>	<code>raw_temperature</code>	Raw temperature value received from sensor

Returns

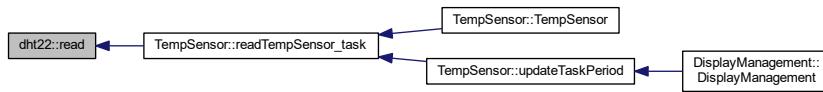
Validity of the read value

Definition at line 30 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.5.4 Member Data Documentation

3.5.4.1 dht22_port

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

Variable containing the port used for 1-wire communication

Definition at line 45 of file dht22.h.

3.5.4.2 dio_ptr

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

Pointer to the DIO object

Definition at line 46 of file dht22.h.

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

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

3.6 dio Class Reference

DIO class.

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

Public Member Functions

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

Private Member Functions

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

Private Attributes

- **uint8_t * PINx_addr_mem**
- **uint8_t PINx_idx_mem**

3.6.1 Detailed Description

DIO class.

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

Definition at line 27 of file dio.h.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 dio()

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

dio class constructor

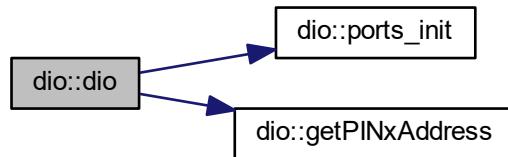
Initializes class dio and calls DIO hardware initialization function

Returns

Nothing

Definition at line 112 of file dio.cpp.

Here is the call graph for this function:



3.6.3 Member Function Documentation

3.6.3.1 dio_changePortPinCnf()

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

Changes the IO configuration of the selected pin.

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

Parameters

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

Returns

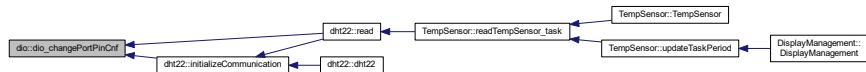
Nothing

Definition at line 149 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.6.3.2 dio_getPort()

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

Gets the logical state of selected pin.

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

Parameters

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

Returns

Logical state of selected pin

Definition at line 139 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.6.3.3 dio_getPort_fast()**

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

Gets the logical state of the memorized pin.

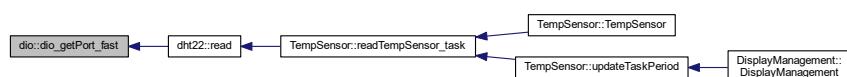
This function gets the logical value of the memorized pin. The corresponding port and pin index are stored in class members PINx_addr_mem and PINx_idx_mem. This mechanism is used to speed up reading time as this function no longer needs to extract register address and pin index from portcode.

Returns

Logical state of selected pin

Definition at line 171 of file dio.cpp.

Here is the caller graph for this function:



3.6.3.4 dio_invertPort()

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

Inverts the state of output port.

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

Parameters

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

Returns

Nothing

Definition at line 131 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.6.3.5 dio_memorizePINaddress()

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

Memorizes PINx register address and pin index.

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

Parameters

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

Returns

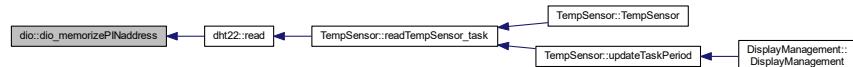
Nothing

Definition at line 165 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.6.3.6 dio_setPort()**

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

Port setting function.

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

Parameters

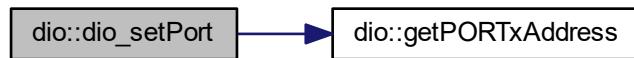
in	<i>portcode</i>	Encoded pin and register index
in	<i>state</i>	Requested state to set pin

Returns

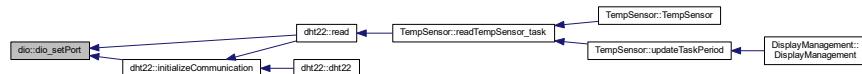
Nothing

Definition at line 121 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.6.3.7 getDDRxAddress()**

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

Gets the physical address of the requested register DDRx.

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

Parameters

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

Returns

Pointer to the DDRx register

Definition at line 83 of file dio.cpp.

Here is the caller graph for this function:



3.6.3.8 getPINxAddress()

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

Gets the physical address of the requested register PINx.

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

Parameters

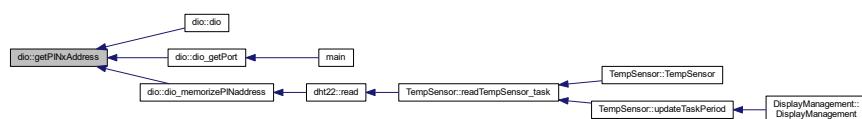
in	<i>portcode</i>	Encoded port code
----	-----------------	-------------------

Returns

Pointer to the PINx register

Definition at line 54 of file dio.cpp.

Here is the caller graph for this function:



3.6.3.9 getPORTxAddress()

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

Gets the physical address of the requested register PORTx.

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

Parameters

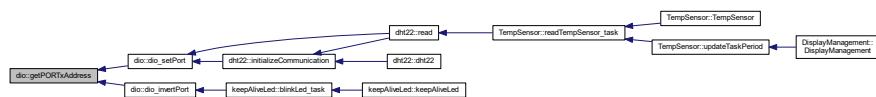
in	<i>portcode</i>	Encoded port code
----	-----------------	-------------------

Returns

Pointer to the PORTx register

Definition at line 25 of file dio.cpp.

Here is the caller graph for this function:



3.6.3.10 ports_init()

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

Digital ports hardware initialization function.

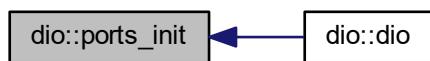
This function initializes digital ports as input or output and sets their initial values

Returns

Nothing

Definition at line 16 of file dio.cpp.

Here is the caller graph for this function:



3.6.4 Member Data Documentation

3.6.4.1 PINx_addr_mem

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

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

Definition at line 142 of file dio.h.

3.6.4.2 PINx_idx_mem

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

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

Definition at line 143 of file dio.h.

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

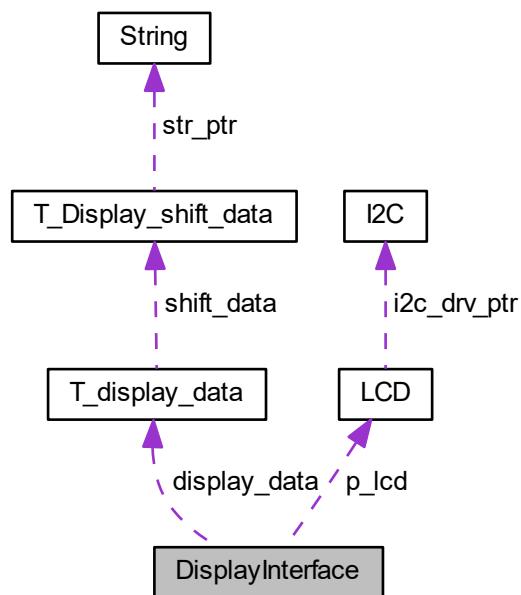
- [dio.h](#)
- [dio.cpp](#)

3.7 DisplayInterface Class Reference

Display interface services class.

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

Collaboration diagram for DisplayInterface:



Public Member Functions

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

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

Line display function.
- bool `ClearLine` (`uint8_t line`)

Line clearing function.
- void `ClearFullScreen` ()

Screen cleaning function.
- bool `IsLineEmpty` (`uint8_t line`)

Empty line get function.
- `T_display_data * getDisplayDataPtr` ()

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

Text alignment function.

Static Public Member Functions

- static void `shiftLine_task` ()

Line shifting periodic task.

Private Member Functions

- `uint8_t FindFirstCharAddr` (`uint8_t line`)

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

Line refresh function.
- void `ClearStringInDataStruct` (`uint8_t line`)

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

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

Line data string update function.

Private Attributes

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

3.7.1 Detailed Description

Display interface services class.

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

Definition at line 76 of file `DisplayInterface.h`.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 DisplayInterface()

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

Class constructor.

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

Parameters

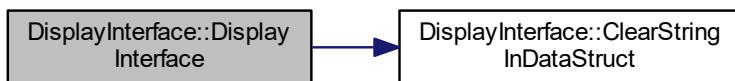
in	<i>LCD_init_cnf</i>	Initial configuration of the screen
----	---------------------	-------------------------------------

Returns

Nothing

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

Here is the call graph for this function:



3.7.3 Member Function Documentation

3.7.3.1 ClearFullScreen()

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

Screen cleaning function.

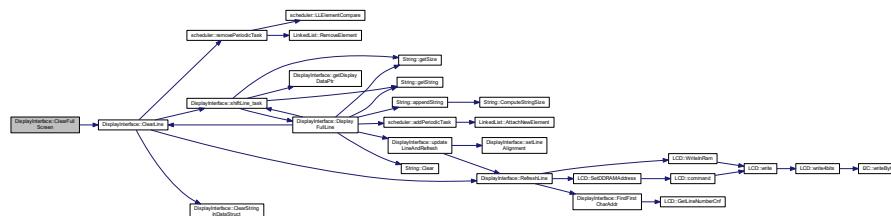
This functions clears the entire display. It uses the `ClearLine` function on every line of the screen.

Returns

Nothing

Definition at line 265 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.3.2 ClearLine()

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

Line clearing function.

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

Parameters

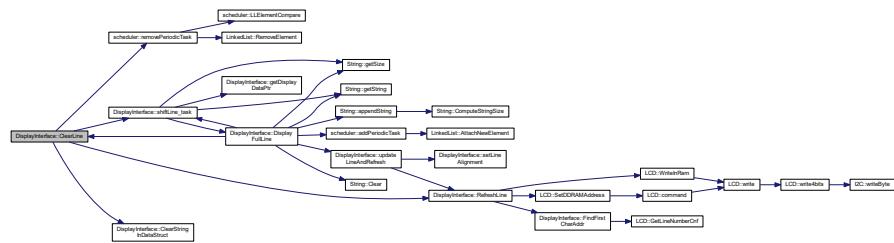
in	<i>line</i>	Line to clear
----	-------------	---------------

Returns

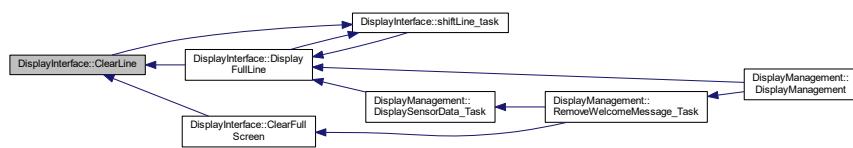
True if the line has been cleared, false otherwise

Definition at line 221 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.3.3 ClearStringInDataStruct()

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

String data clearing structure.

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

Parameters

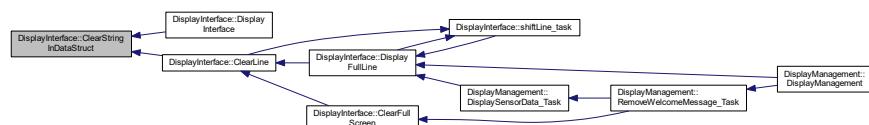
in	<i>line</i>	Line to clear
----	-------------	---------------

Returns

Nothing

Definition at line 174 of file DisplayInterface.cpp.

Here is the caller graph for this function:



3.7.3.4 DisplayFullLine()

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

Line display function.

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

Parameters

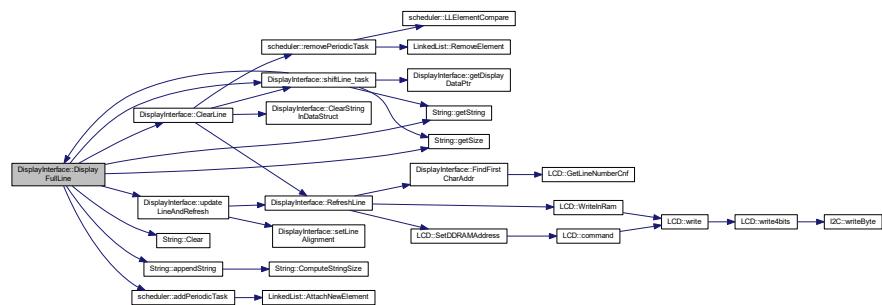
in	str	Pointer to the string to display
in	size	Size of the string to display
in	line	Index of the line where the string shall be displayed
in	mode	Display mode
in	alignment	Requested alignment for the line

Returns

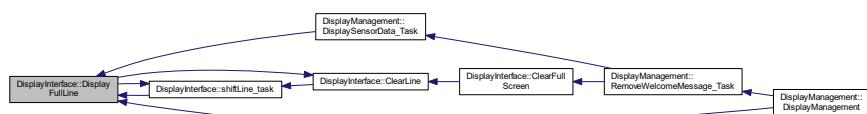
True if the line has been correctly displayed, false otherwise

Definition at line 59 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.3.5 FindFirstCharAddr()

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

Finds start address of a line.

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

Parameters

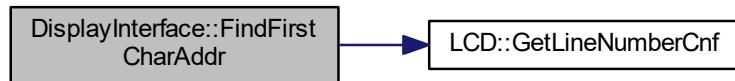
<code>in</code>	<code>line</code>	Line which address shall be found
-----------------	-------------------	-----------------------------------

Returns

Address in DDRAM of the first character of the line

Definition at line 182 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.3.6 getDisplayDataPtr()

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

Display data get function.

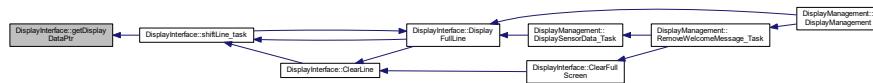
This function returns a pointer to the display data structure.

Returns

Pointer to display data structure.

Definition at line 142 of file DisplayInterface.h.

Here is the caller graph for this function:

**3.7.3.7 IsLineEmpty()**

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

Empty line get function.

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

Parameters

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

Returns

True if the line is empty, false otherwise

Definition at line 273 of file DisplayInterface.cpp.

3.7.3.8 RefreshLine()

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

Line refresh function.

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

Parameters

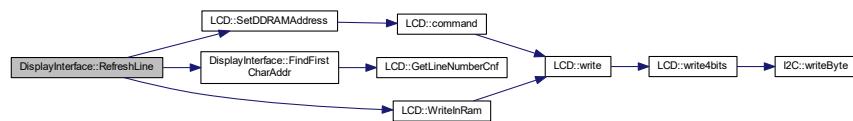
in	<i>line</i>	Line to refresh
----	-------------	-----------------

Returns

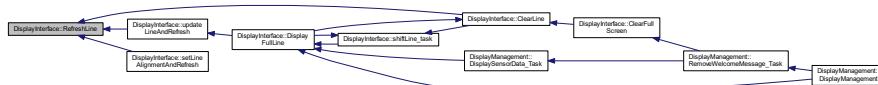
Nothing

Definition at line 161 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.7.3.9 setLineAlignment()**

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

Text alignment setting function.

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

Parameters

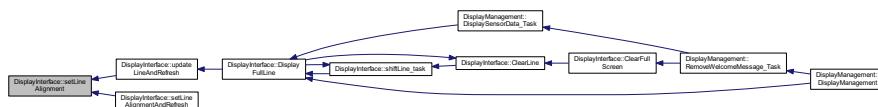
in	<i>line</i>	Line to update
----	-------------	----------------

Returns

Nothing

Definition at line 330 of file DisplayInterface.cpp.

Here is the caller graph for this function:



3.7.3.10 setLineAlignmentAndRefresh()

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

Text alignment function.

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

Parameters

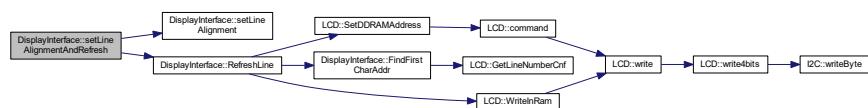
in	<i>line</i>	Requested line to update
in	<i>alignment</i>	Requested alignment for the text

Returns

Nothing

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

Here is the call graph for this function:



3.7.3.11 shiftLine_task()

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

Line shifting periodic task.

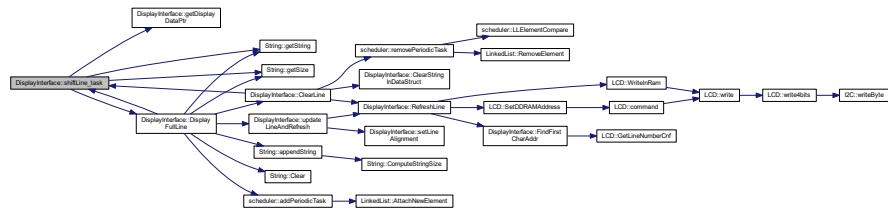
This function is called periodically by the scheduler. It shifts all the lines in line shifting mode and updates the data structures.

Returns

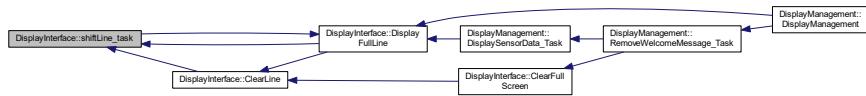
Nothing

Definition at line 282 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.3.12 updateLineAndRefresh()

```
void DisplayInterface::updateLineAndRefresh (
```

	uint8_t * str,
	uint8_t size,
	uint8_t line) [private]

Line data string update function.

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

Parameters

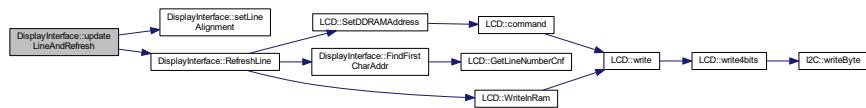
in	<i>str</i>	Pointer to the string to display
in	<i>size</i>	Size of the string
in	<i>line</i>	Line to update

Returns

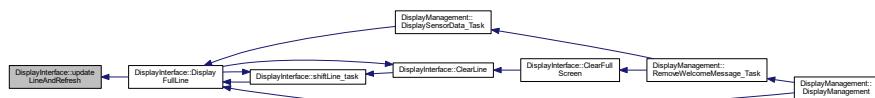
Nothing

Definition at line 145 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.4 Member Data Documentation

3.7.4.1 display_data

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

Screen display data

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

3.7.4.2 dummy

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

Needed for data alignment

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

3.7.4.3 isShiftInProgress

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

Flag indicating if a shift is in progress on any line

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

3.7.4.4 p_lcd

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

Pointer to the attached LCD driver object

Definition at line 163 of file DisplayInterface.h.

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

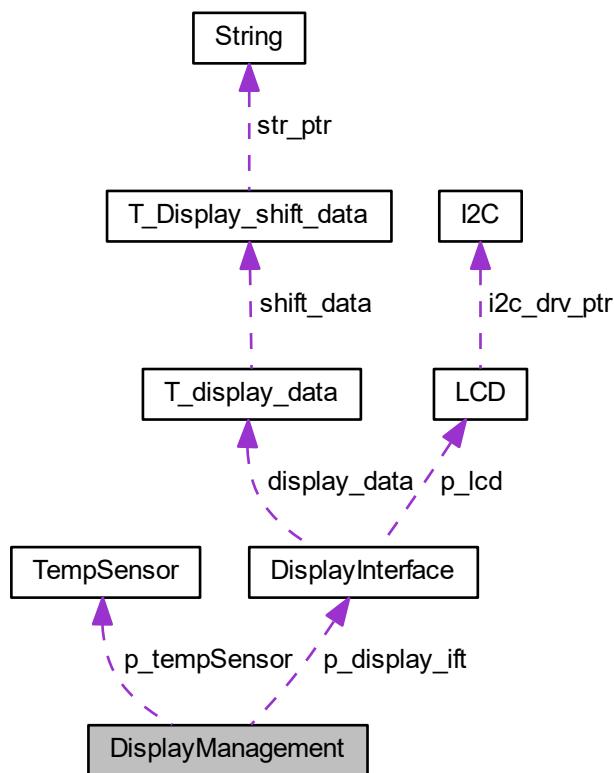
- [DisplayInterface.h](#)
- [DisplayInterface.cpp](#)

3.8 DisplayManagement Class Reference

Display management class.

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

Collaboration diagram for DisplayManagement:



Public Member Functions

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

Static Public Member Functions

- [static void DisplaySensorData_Task \(\)](#)
Periodic task for displaying sensor data.
- [static void RemoveWelcomeMessage_Task \(\)](#)
End of welcome message task.

Private Attributes

- [DisplayInterface * p_display_ift](#)
- [TempSensor * p_tempSensor](#)

3.8.1 Detailed Description

Display management class.

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

Definition at line 53 of file [DisplayManagement.h](#).

3.8.2 Constructor & Destructor Documentation

3.8.2.1 [DisplayManagement\(\)](#)

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

Class constructor.

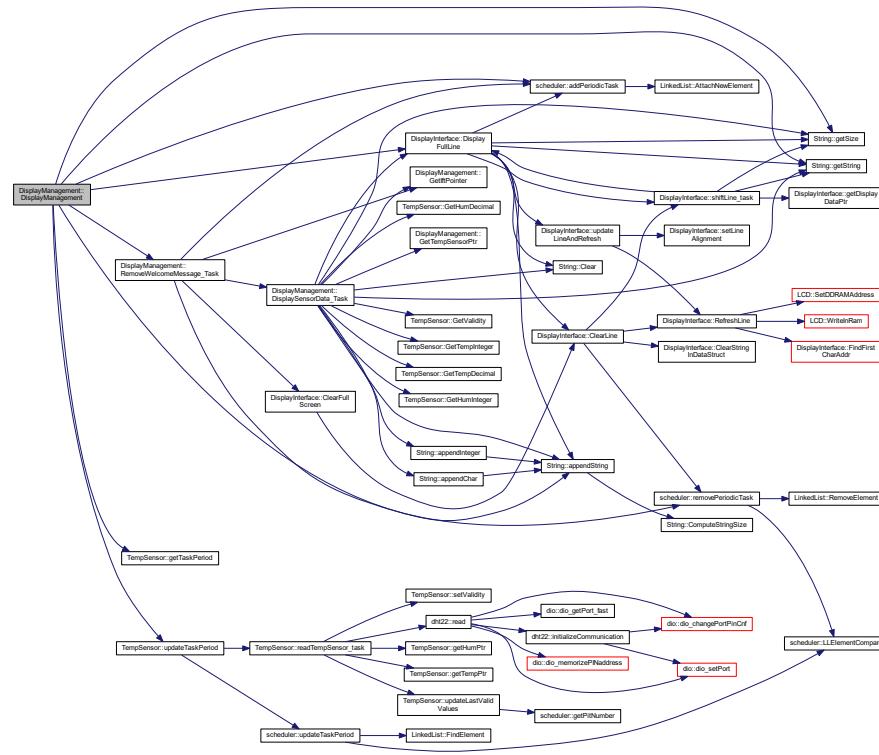
This class initializes display management.
It creates a display interface object and initializes all class variables.

Returns

Nothing

Definition at line 30 of file DisplayManagement.cpp.

Here is the call graph for this function:



3.8.3 Member Function Documentation

3.8.3.1 `DisplaySensorData_Task()`

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

Periodic task for displaying sensor data.

This function displays the sensors data on the screen. Currently temperature and humidity data coming from `dht22` sensor are displayed.

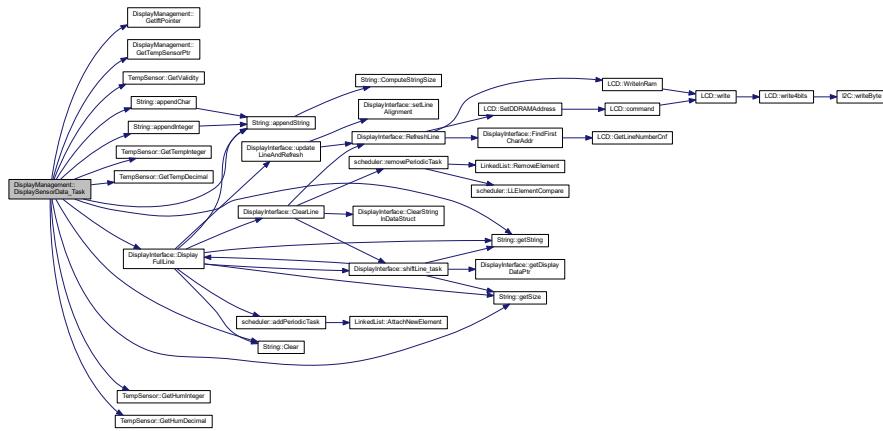
It is called periodically by scheduler.

Returns

Nothing

Definition at line 74 of file DisplayManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.8.3.2 GetIftPointer()**

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

Interface pointer get function.

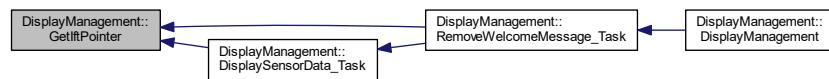
This function returns the pointer to the display interface object

Returns

Pointer to display interface object

Definition at line 82 of file DisplayManagement.h.

Here is the caller graph for this function:



3.8.3.3 GetTempSensorPtr()

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

Sensor pointer get function.

This function returns the pointer to the temperature sensor object

Returns

Pointer to sensor object

Definition at line 93 of file DisplayManagement.h.

Here is the caller graph for this function:



3.8.3.4 RemoveWelcomeMessage_Task()

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

End of welcome message task.

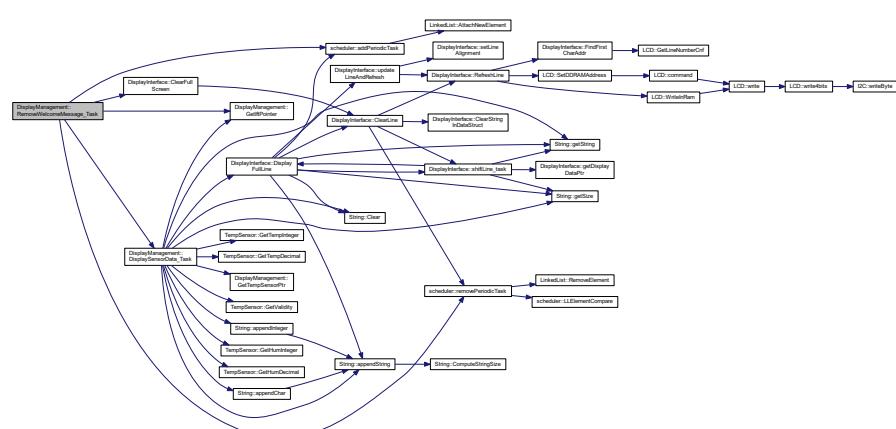
This task clears the welcome message from the screen and start periodic display of sensor data. This task shall be added in scheduler when the welcome message is displayed on screen. As it shall be called only once, the task removes itself from the scheduler after the first call.

Returns

Nothing

Definition at line 57 of file DisplayManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.8.4 Member Data Documentation

3.8.4.1 p_display_ift

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

Pointer to the display interface object

Definition at line 110 of file DisplayManagement.h.

3.8.4.2 p_tempSensor

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

Pointer to the temperature sensor object

Definition at line 111 of file DisplayManagement.h.

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

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

3.9 I2C Class Reference

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

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

Public Member Functions

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

Private Member Functions

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

Private Attributes

- `uint8_t tx_address`
 - `uint32_t bitrate`

3.9.1 Detailed Description

Two-wire serial interface (I2C) class definition.

This class manages I₂C driver.

Definition at line 23 of file I2C.h.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 I2C()

```
I2C:::I2C (
```

I2C class constructor.

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

Parameters

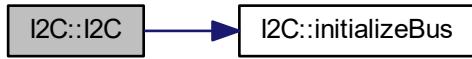
in	<i>I_bitrate</i>	Requested bitrate for I ₂ C bus (in Hz)
----	------------------	--

Returns

Nothing

Definition at line 16 of file I2C.cpp.

Here is the call graph for this function:



3.9.3 Member Function Documentation

3.9.3.1 initializeBus()

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

I2C bus initialization.

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

Returns

Nothing

Definition at line 77 of file I2C.cpp.

Here is the caller graph for this function:



3.9.3.2 setBitRate()

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

Variable bitrate setting function.

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

Parameters

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

Returns

Nothing

Definition at line 72 of file I2C.cpp.

3.9.3.3 setTxAddress()

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

Setting function for Tx **I2C** address.

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

Parameters

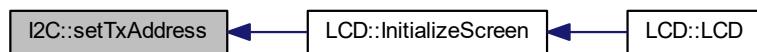
in	<i>address</i>	Requested Tx address
----	----------------	----------------------

Returns

Nothing

Definition at line 67 of file I2C.cpp.

Here is the caller graph for this function:

**3.9.3.4 writeByte()**

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

Byte sending function.

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

Parameters

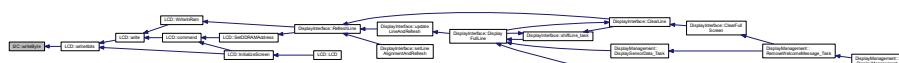
in	<i>data</i>	Pointer to the data to send
----	-------------	-----------------------------

Returns

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

Definition at line 24 of file I2C.cpp.

Here is the caller graph for this function:



3.9.4 Member Data Documentation

3.9.4.1 bitrate

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

Definition at line 63 of file I2C.h.

3.9.4.2 tx_address

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

Definition at line 62 of file I2C.h.

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

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

3.10 keepAliveLed Class Reference

Class for keep-alive LED blinking.

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

Public Member Functions

- [keepAliveLed \(\)](#)

Class constructor.

Static Public Member Functions

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

Task for LED blinking.

3.10.1 Detailed Description

Class for keep-alive LED blinking.

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

Definition at line 22 of file keepAliveLed.h.

3.10.2 Constructor & Destructor Documentation

3.10.2.1 keepAliveLed()

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

Class constructor.

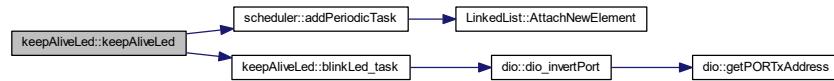
This function initializes the class keepAliveLed

Returns

Nothing

Definition at line 22 of file keepAliveLed.cpp.

Here is the call graph for this function:



3.10.3 Member Function Documentation

3.10.3.1 blinkLed_task()

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

Task for LED blinking.

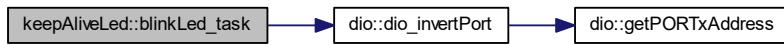
This function is inserted into the scheduler. It changes the state of the LED output to make it blink

Returns

Nothing

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

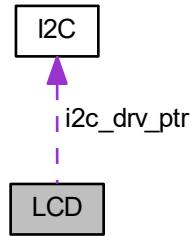
- [keepAliveLed.h](#)
- [keepAliveLed.cpp](#)

3.11 LCD Class Reference

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

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

Collaboration diagram for LCD:



Public Member Functions

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

Private Member Functions

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

Private Attributes

- bool `backlight_enable`
- bool `cnfLineNumber`
- bool `cnfFontType`
- bool `cnfDisplayOnOff`
- bool `cnfCursorOnOff`
- bool `cnfCursorBlink`
- bool `cnfEntryModeDir`
- bool `cnfEntryModeShift`
- uint8_t `cnfI2C_addr`
- I2C * `i2c_drv_ptr`
- uint8_t `ddram_addr`

3.11.1 Detailed Description

Class for `LCD` S2004A display driver.

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

Definition at line 147 of file LCD.h.

3.11.2 Constructor & Destructor Documentation

3.11.2.1 LCD()

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

`LCD` class constructor.

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

Parameters

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

Returns

Nothing

Definition at line 18 of file LCD.cpp.

Here is the call graph for this function:



3.11.3 Member Function Documentation

3.11.3.1 command()

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

LCD command management function.

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

Parameters

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

Returns

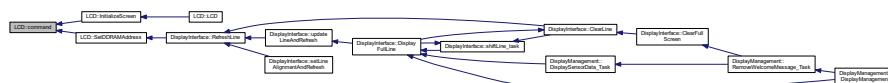
Nothing

Definition at line 125 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.11.3.2 ConfigureBacklight()

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

Backlight configuration function.

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

Parameters

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

Returns

Nothing

Definition at line 178 of file LCD.h.

Here is the caller graph for this function:



3.11.3.3 ConfigureCursorBlink()

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

Cursor blinking configuration function.

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

Parameters

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

Returns

Nothing

Definition at line 238 of file LCD.h.

Here is the caller graph for this function:



3.11.3.4 ConfigureCursorOnOff()

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

Cursor configuration function.

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

Parameters

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

Returns

Nothing

Definition at line 226 of file LCD.h.

Here is the caller graph for this function:



3.11.3.5 ConfigureDisplayOnOff()

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

Display configuration function.

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

Parameters

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

Returns

Nothing

Definition at line 214 of file LCD.h.

Here is the caller graph for this function:



3.11.3.6 ConfigureEntryModeDir()

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

Entry mode direction configuration function.

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

Parameters

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

Returns

Nothing

Definition at line 250 of file LCD.h.

Here is the caller graph for this function:



3.11.3.7 ConfigureEntryModeShift()

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

Entry mode shift configuration function.

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

Parameters

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

Returns

Nothing

Definition at line 262 of file LCD.h.

Here is the caller graph for this function:



3.11.3.8 ConfigureFontType()

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

Font configuration function.

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

Parameters

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

Returns

Nothing

Definition at line 202 of file LCD.h.

Here is the caller graph for this function:



3.11.3.9 ConfigureI2CAddr()

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

I2C address configuration function.

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

Parameters

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

Returns

Nothing

Definition at line 274 of file LCD.h.

Here is the caller graph for this function:



3.11.3.10 ConfigureLineNumber()

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

Line type configuration function.

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

Parameters

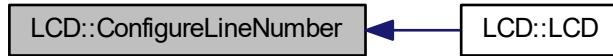
in	param	Configuration value
----	-------	---------------------

Returns

Nothing

Definition at line 190 of file LCD.h.

Here is the caller graph for this function:



3.11.3.11 GetDDRAMAddress()

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

DDRAM address get function.

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

Returns

Current DDRAM address

Definition at line 294 of file LCD.h.

3.11.3.12 GetLineNumberCnf()

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

Number of line get function.

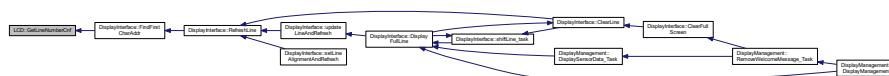
This function returns the line number configuration of the screen : 1 or 2 lines mode.

Returns

Line number configuration

Definition at line 316 of file LCD.h.

Here is the caller graph for this function:



3.11.3.13 InitializeScreen()

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

Screen configuration function.

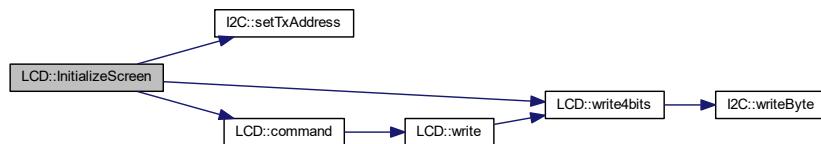
This function configures the [LCD](#) screen. It's must be called during initialization phase, or the screen won't be usable. The configuration process is described in [LCD](#) datasheet J2004A-GFDN-DYNC

Returns

Nothing

Definition at line 73 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.11.3.14 SetDDRAMAddress()

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

DDRAM address setting function.

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

Parameters

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

Returns

Nothing

Definition at line 168 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.11.3.15 write()**

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

I2C write function.

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

Parameters

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

Returns

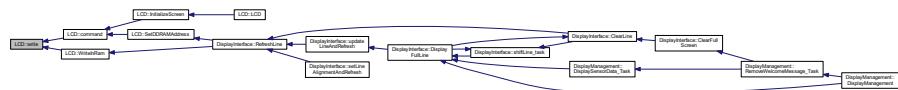
Nothing

Definition at line 62 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.11.3.16 write4bits()

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

I2C write function for 4-bits mode.

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

Parameters

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

Returns

Nothing

Definition at line 45 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.11.3.17 WriteInRam()

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

Screen RAM write function.

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

Parameters

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

Returns

Nothing

Definition at line 190 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.11.4 Member Data Documentation

3.11.4.1 `backlight_enable`

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

Backlight enable flag

Definition at line 324 of file LCD.h.

3.11.4.2 `cnfCursorBlink`

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

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

Definition at line 329 of file LCD.h.

3.11.4.3 `cnfCursorOnOff`

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

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

Definition at line 328 of file LCD.h.

3.11.4.4 `cnfDisplayOnOff`

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

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

Definition at line 327 of file LCD.h.

3.11.4.5 `cnfEntryModeDir`

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

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

Definition at line 330 of file LCD.h.

3.11.4.6 cnfEntryModeShift

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

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

Definition at line 331 of file LCD.h.

3.11.4.7 cnfFontType

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

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

Definition at line 326 of file LCD.h.

3.11.4.8 cnfI2C_addr

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

I2C address of the [LCD](#) screen

Definition at line 332 of file LCD.h.

3.11.4.9 cnfLineNumber

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

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

Definition at line 325 of file LCD.h.

3.11.4.10 ddram_addr

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

Screen DDRAM address

Definition at line 336 of file LCD.h.

3.11.4.11 i2c_drv_ptr

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

Pointer to the [I2C](#) driver object

Definition at line 334 of file LCD.h.

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

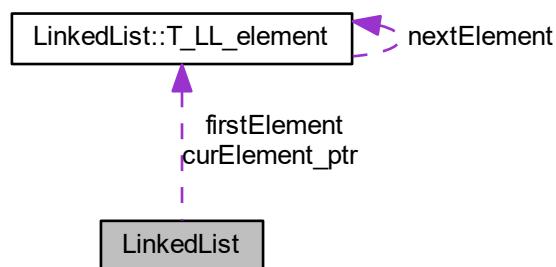
- [LCD.h](#)
- [LCD.cpp](#)

3.12 LinkedList Class Reference

Linked list class.

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

Collaboration diagram for LinkedList:



Classes

- struct [T_LL_element](#)

Type defining a linked list element.

Public Member Functions

- [LinkedList \(\)](#)
Class constructor.
- [~LinkedList \(\)](#)
Class destructor.
- [void AttachNewElement \(void *data_ptr\)](#)
Add an new element to the list.
- [bool RemoveElement \(CompareFctPtr_t comparisonFct_ptr, void *reference_ptr\)](#)
Removes an element from the chain.
- [void *getCurrentElement \(\)](#)
Current element get function.
- [bool MoveToNextElement \(\)](#)
Move to next element function.
- [void ResetElementPtr \(\)](#)
Resets element pointer.
- [bool IsLLEmpty \(\)](#)
Empty linked list.
- [bool FindElement \(CompareFctPtr_t comparisonFct_ptr, void *reference_ptr, void **chainElement_ptr\)](#)
Element finding function.

Private Types

- [typedef struct LinkedList::T_LL_element T_LL_element](#)
Type defining a linked list element.

Private Attributes

- [T_LL_element * firstElement](#)
- [T_LL_element * curElement_ptr](#)

3.12.1 Detailed Description

Linked list class.

This class defines a linked list and the associated services.

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

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

3.12.2 Member Typedef Documentation

3.12.2.1 T_LL_element

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

Type defining a linked list element.

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

3.12.3 Constructor & Destructor Documentation

3.12.3.1 LinkedList()

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

Class constructor.

This constructor initializes a linked list

Returns

Nothing

Definition at line 37 of file LinkedList.cpp.

3.12.3.2 ~LinkedList()

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

Class destructor.

This function deletes the linked list

Returns

Nothing

Definition at line 43 of file LinkedList.cpp.

Here is the call graph for this function:



3.12.4 Member Function Documentation

3.12.4.1 AttachNewElement()

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

Add an new element to the list.

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

Parameters

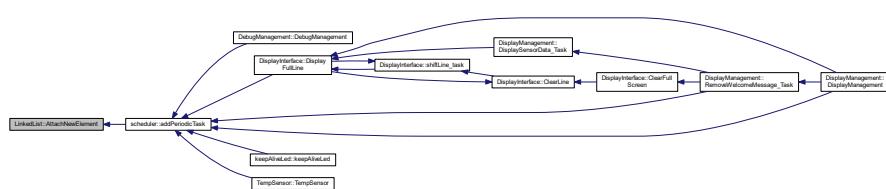
in	<i>data_ptr</i>	Pointer to the data element
----	-----------------	-----------------------------

Returns

Nothing

Definition at line 61 of file LinkedList.cpp.

Here is the caller graph for this function:



3.12.4.2 FindElement()

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

Element finding function.

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

Parameters

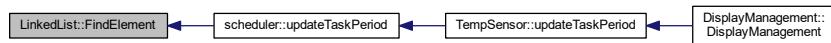
in	<i>comparisonFct_ptr</i>	Pointer to the comparison function
in	<i>reference_ptr</i>	Pointer to the element to find in the chain
out	<i>chainElement_ptr</i>	Pointer to pointer to the found element

Returns

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

Definition at line 152 of file LinkedList.cpp.

Here is the caller graph for this function:

**3.12.4.3 getCurrentElement()**

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

Current element get function.

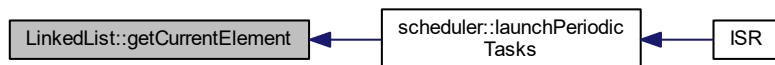
This function returns a pointer to the current pointed data in the chain.

Returns

Pointer to the current data

Definition at line 67 of file LinkedList.h.

Here is the caller graph for this function:



3.12.4.4 IsLLEmpty()

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

Empty linked list.

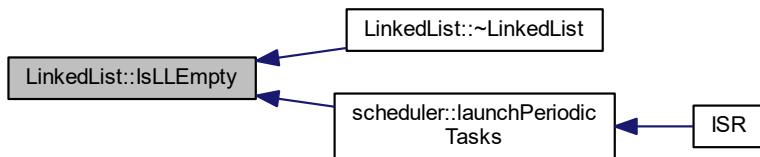
This function checks whether the linked list is empty or not (pointer to first element is equal to 0 or not).

Returns

True if the list is empty, false otherwise

Definition at line 144 of file LinkedList.cpp.

Here is the caller graph for this function:



3.12.4.5 MoveToNextElement()

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

Move to next element function.

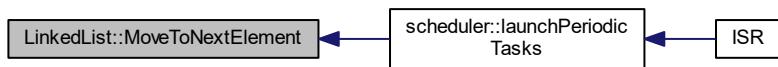
This function moves the element pointer to the next element of the chain.

Returns

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

Definition at line 130 of file LinkedList.cpp.

Here is the caller graph for this function:



3.12.4.6 RemoveElement()

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

Removes an element from the chain.

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

Parameters

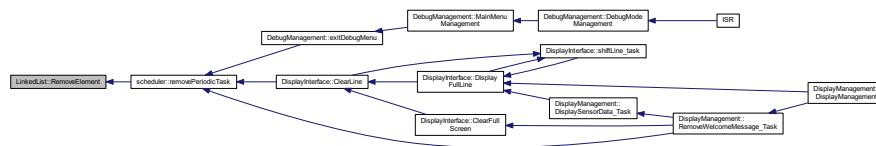
in	<i>comparisonFct_ptr</i>	Pointer to the comparison function to use
in	<i>reference_ptr</i>	Pointer to the reference data used for comparison

Returns

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

Definition at line 86 of file LinkedList.cpp.

Here is the caller graph for this function:



3.12.4.7 ResetElementPtr()

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

Resets element pointer.

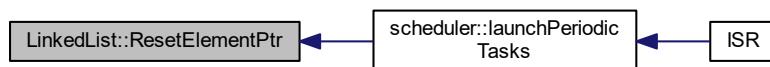
This function sets the element pointer to the first element of the chain.

Returns

Nothing

Definition at line 86 of file LinkedList.h.

Here is the caller graph for this function:



3.12.5 Member Data Documentation

3.12.5.1 curElement_ptr

`T_LL_element* LinkedList::curElement_ptr [private]`

Pointer to the current element of the list

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

3.12.5.2 firstElement

`T_LL_element* LinkedList::firstElement [private]`

Pointer to the first element of the list

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

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

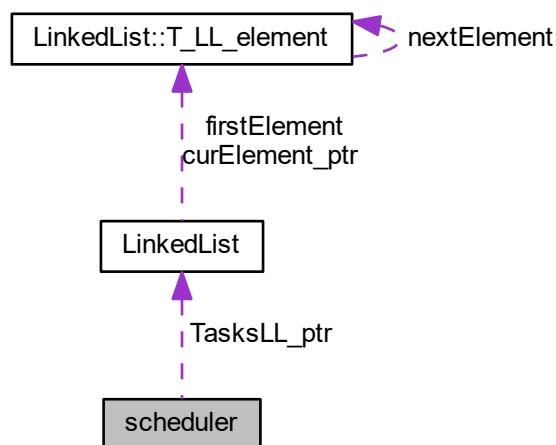
- [LinkedList.h](#)
- [LinkedList.cpp](#)

3.13 scheduler Class Reference

Scheduler class.

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

Collaboration diagram for scheduler:



Classes

- struct [Task_t](#)
Type defining a task structure.

Public Member Functions

- [scheduler \(\)](#)
scheduler class constructor
- void [launchPeriodicTasks \(\)](#)
Main scheduler function.
- void [startScheduling \(\)](#)
Starts the tasks scheduling.
- void [addPeriodicTask \(TaskPtr_t task_ptr, uint16_t a_period\)](#)
Add a task into the scheduler.
- bool [removePeriodicTask \(TaskPtr_t task_ptr\)](#)
Remove a task from the scheduler.
- uint32_t [getPitNumber \(\)](#)
Get function for PIT number.
- bool [updateTaskPeriod \(TaskPtr_t task_ptr, uint16_t period\)](#)
Task period update function.
- uint8_t [getTaskCount \(\)](#)
Task count get function.

Static Public Member Functions

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

Private Types

- typedef struct [scheduler::Task_t Task_t](#)
Type defining a task structure.

Private Attributes

- uint8_t [task_count](#)
- [LinkedList * TasksLL_ptr](#)
- uint32_t [pit_number](#)

3.13.1 Detailed Description

Scheduler class.

This class defines the scheduler of the system.

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

Definition at line 30 of file scheduler.h.

3.13.2 Member Typedef Documentation

3.13.2.1 Task_t

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

Type defining a task structure.

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

3.13.3 Constructor & Destructor Documentation

3.13.3.1 scheduler()

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

scheduler class constructor

This function initializes the class scheduler

Returns

Nothing

Definition at line 29 of file scheduler.cpp.

Here is the call graph for this function:



3.13.4 Member Function Documentation

3.13.4.1 addPeriodicTask()

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

Add a task into the scheduler.

This function create a new task in the scheduler linked to the function task_ptr with a period a_period and an ID a_task_id

Parameters

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

Returns

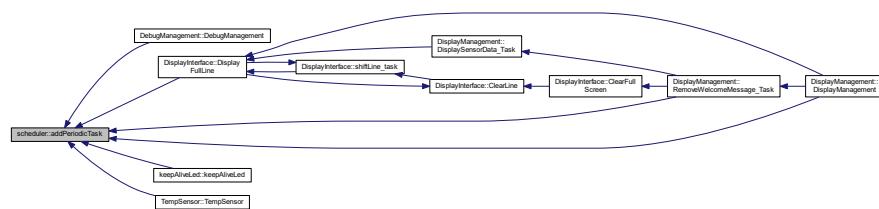
Nothing

Definition at line 99 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.13.4.2 getPitNumber()**

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

Get function for PIT number.

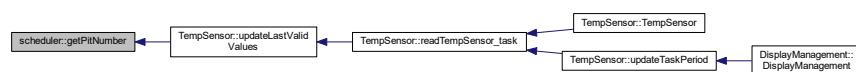
This function returns the PIT number

Returns

PIT number

Definition at line 113 of file scheduler.cpp.

Here is the caller graph for this function:



3.13.4.3 getTaskCount()

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

Task count get function.

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

Returns

Number of tasks

Definition at line 115 of file scheduler.h.

3.13.4.4 launchPeriodicTasks()

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

Main scheduler function.

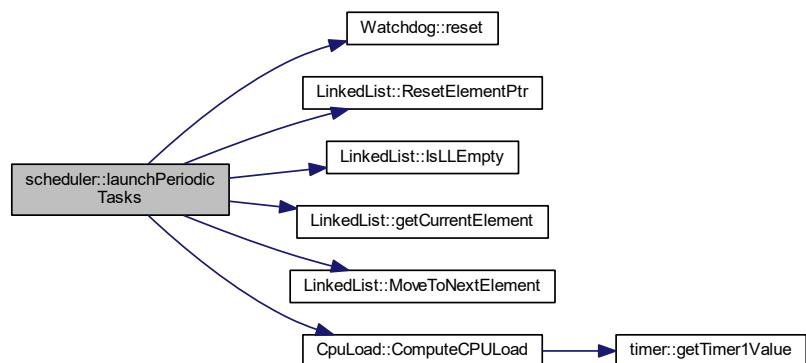
This function launches the scheduled tasks according to current software time and task configuration

Returns

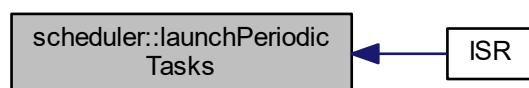
Nothing

Definition at line 54 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.13.4.5 LLElementCompare()

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

Linked list comparison function.

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

Parameters

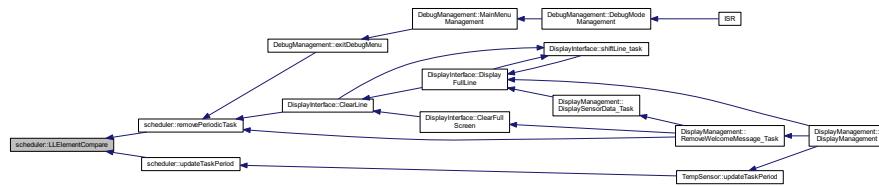
in	<i>LLElement</i>	Pointer to the linked list element
in	<i>CompareElement</i>	Pointer to the element to the compare

Returns

True if both elements are identical, false otherwise

Definition at line 131 of file scheduler.cpp.

Here is the caller graph for this function:



3.13.4.6 removePeriodicTask()

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

Remove a task from the scheduler.

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

Parameters

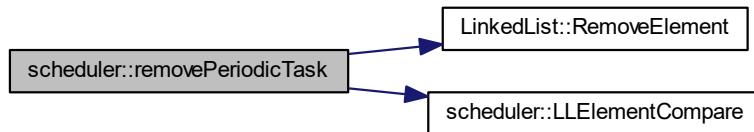
in	<i>task_ptr</i>	address of the task to remove from scheduler
----	-----------------	--

Returns

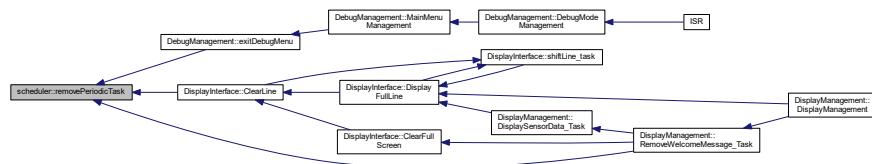
TRUE if the task has been removed, FALSE if the task does not exist in the scheduler

Definition at line 119 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.13.4.7 startScheduling()**

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

Starts the tasks scheduling.

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

Returns

Nothing

Definition at line 93 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.13.4.8 updateTaskPeriod()

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

Task period update function.

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

Parameters

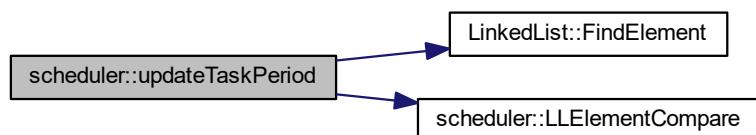
in	<i>task_ptr</i>	Pointer of the task to update
in	<i>period</i>	New period of the task

Returns

True if the update has been correctly done, false otherwise

Definition at line 142 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.13.5 Member Data Documentation

3.13.5.1 pit_number

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

Counter of periodic interrupts

Definition at line 140 of file scheduler.h.

3.13.5.2 task_count

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

Number of task in scheduler

Definition at line 136 of file scheduler.h.

3.13.5.3 TasksLL_ptr

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

Pointer to the linked list object containing the tasks

Definition at line 138 of file scheduler.h.

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

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

3.14 String Class Reference

`String` management class.

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

Public Member Functions

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

Private Member Functions

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

Private Attributes

- `uint8_t * string`
- `uint8_t size`

3.14.1 Detailed Description

`String` management class.

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

Definition at line 18 of file String.h.

3.14.2 Constructor & Destructor Documentation

3.14.2.1 String() [1/2]

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

Class constructor.

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

Parameters

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

Returns

Nothing

Definition at line 15 of file String.cpp.

Here is the call graph for this function:



3.14.2.2 String() [2/2]

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

Class constructor.

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

Returns

Nothing

Definition at line 33 of file String.cpp.

3.14.2.3 ~String()

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

Class destructor.

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

Returns

Nothing

Definition at line 39 of file String.cpp.

Here is the call graph for this function:



3.14.3 Member Function Documentation

3.14.3.1 appendBool()

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

Boolean adding function.

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

Parameters

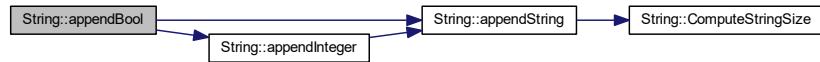
in	<i>data</i>	Boolean data to add
in	<i>isText</i>	Defines the conversion mode : text or integer

Returns

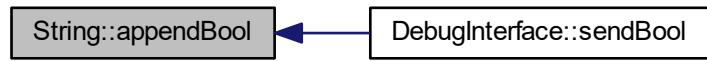
Nothing

Definition at line 121 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.14.3.2 appendChar()

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

Character adding function.

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

Parameters

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

Returns

Nothing

Definition at line 139 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.14.3.3 appendInteger()

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

Integer adding function.

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

Parameters

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

Returns

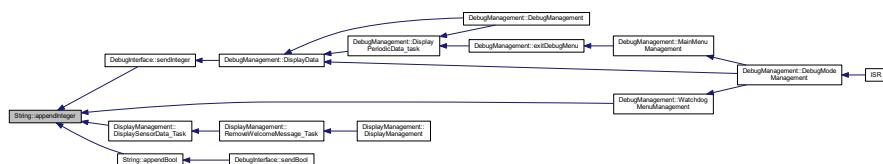
Nothing

Definition at line 95 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.14.3.4 appendString()

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

[String](#) adding function.

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

Parameters

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

Returns

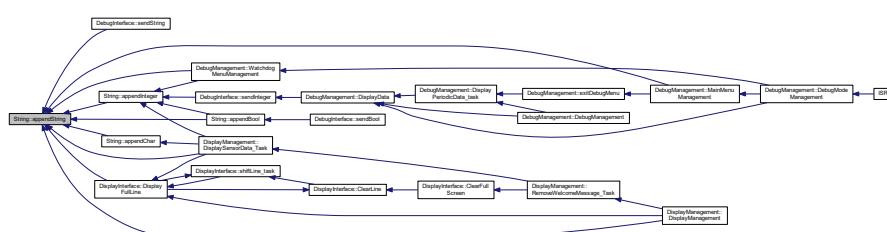
Nothing

Definition at line 57 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.14.3.5 Clear()

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

[String](#) clear function.

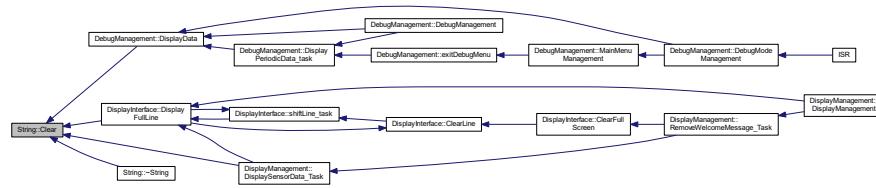
This function clears the string. Size is set to 0 and the memory is freed.

Returns

Nothing

Definition at line 113 of file String.cpp.

Here is the caller graph for this function:

**3.14.3.6 ComputeStringSize()**

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

[String](#) size computation function.

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

Parameters

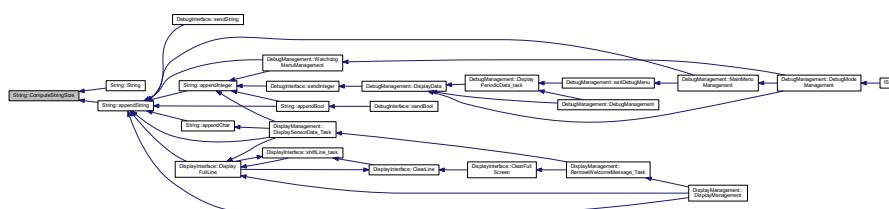
in	str	Pointer to the beginning of the string
----	-----	--

Returns

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

Definition at line 44 of file String.cpp.

Here is the caller graph for this function:



3.14.3.7 getSize()

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

Size get function.

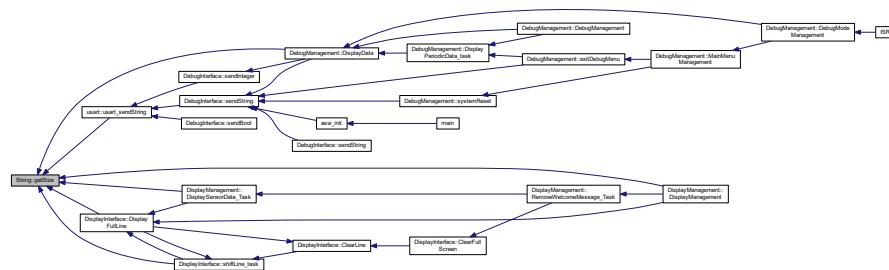
This function returns the size of the string.

Returns

Size of the string

Definition at line 64 of file String.h.

Here is the caller graph for this function:



3.14.3.8 getString()

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

[String](#) pointer get function.

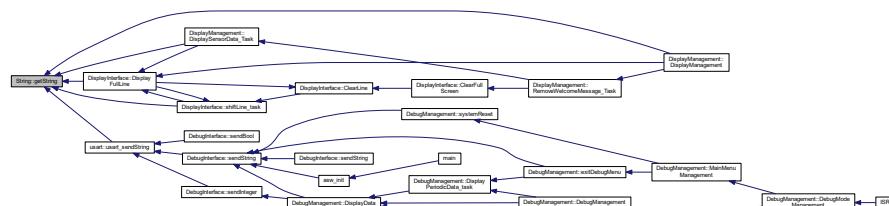
This function returns the pointer to the beginning of the string.

Returns

[String](#) pointer

Definition at line 53 of file String.h.

Here is the caller graph for this function:



3.14.4 Member Data Documentation

3.14.4.1 size

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

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

Definition at line 121 of file String.h.

3.14.4.2 string

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

Pointer to the start of the string

Definition at line 120 of file String.h.

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

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

3.15 T_ASW_init_cnf Struct Reference

ASW initialization configuration structure.

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

Public Attributes

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

3.15.1 Detailed Description

ASW initialization configuration structure.

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

Definition at line 17 of file asw.h.

3.15.2 Member Data Documentation

3.15.2.1 isDebugEnabled

```
bool T_ASW_init_cnf::isDebugEnabled
```

Debug services activation flag

Definition at line 19 of file asw.h.

3.15.2.2 isDisplayActivated

```
bool T_ASW_init_cnf::isDisplayActivated
```

LCD display activation flag

Definition at line 22 of file asw.h.

3.15.2.3 isLEDActivated

```
bool T_ASW_init_cnf::isLEDActivated
```

Keep-alive LED activation flag

Definition at line 20 of file asw.h.

3.15.2.4 isTempSensorActivated

```
bool T_ASW_init_cnf::isTempSensorActivated
```

Temperature sensor activation flag

Definition at line 21 of file asw.h.

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

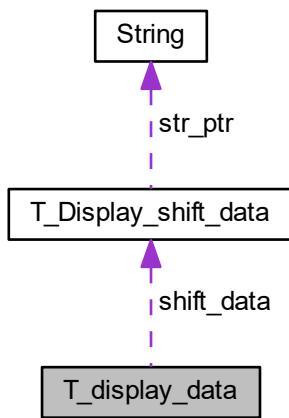
- [asw.h](#)

3.16 T_display_data Struct Reference

Structure containing display data.

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

Collaboration diagram for T_display_data:



Public Attributes

- bool [isEmpty](#)
- [T_DisplayInterface_LineDisplayMode mode](#)
- [T_DisplayInterface_LineAlignment alignment](#)
- [T_Display_shift_data shift_data](#)
- uint8_t [display_str \[LCD_SIZE_NB_CHAR_PER_LINE\]](#)

3.16.1 Detailed Description

Structure containing display data.

This structure contains all data used for screen display

Definition at line 57 of file [DisplayInterface.h](#).

3.16.2 Member Data Documentation

3.16.2.1 alignment

```
T_DisplayInterface_LineAlignment T_display_data::alignment
```

Line alignment

Definition at line 61 of file DisplayInterface.h.

3.16.2.2 display_str

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

Current string displayed on the screen

Definition at line 63 of file DisplayInterface.h.

3.16.2.3 isEmpty

```
bool T_display_data::isEmpty
```

Flag indicating if the line is empty or not

Definition at line 59 of file DisplayInterface.h.

3.16.2.4 mode

```
T_DisplayInterface_LineDisplayMode T_display_data::mode
```

Current display mode

Definition at line 60 of file DisplayInterface.h.

3.16.2.5 shift_data

```
T_Display_shift_data T_display_data::shift_data
```

Shift data for the current line

Definition at line 62 of file DisplayInterface.h.

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

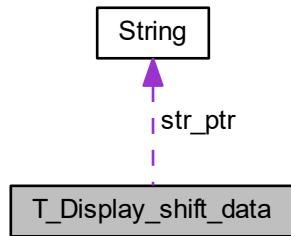
- [DisplayInterface.h](#)

3.17 T_Display_shift_data Struct Reference

Structure containing shift data.

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

Collaboration diagram for T_Display_shift_data:



Public Attributes

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

3.17.1 Detailed Description

Structure containing shift data.

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

Definition at line 45 of file DisplayInterface.h.

3.17.2 Member Data Documentation

3.17.2.1 str_cur_ptr

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

Pointer to the address of the first displayed character

Definition at line 48 of file DisplayInterface.h.

3.17.2.2 str_ptr

`String* T_Display_shift_data::str_ptr`

Pointer to the start address of the string

Definition at line 47 of file DisplayInterface.h.

3.17.2.3 temporization

`uint8_t T_Display_shift_data::temporization`

Shifting period

Definition at line 49 of file DisplayInterface.h.

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

- [DisplayInterface.h](#)

3.18 T_LCD_conf_struct Struct Reference

Structure defining [LCD](#) configuration.

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

Public Attributes

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

3.18.1 Detailed Description

Structure defining [LCD](#) configuration.

Definition at line 128 of file LCD.h.

3.18.2 Member Data Documentation

3.18.2.1 `backlight_en`

```
bool T_LCD_conf_struct::backlight_en
```

Screen backlight enable flag

Definition at line 132 of file LCD.h.

3.18.2.2 `cursor_en`

```
bool T_LCD_conf_struct::cursor_en
```

Screen cursor enable flag

Definition at line 136 of file LCD.h.

3.18.2.3 `cursorBlink_en`

```
bool T_LCD_conf_struct::cursorBlink_en
```

Screen cursor blinking enable flag

Definition at line 137 of file LCD.h.

3.18.2.4 `display_en`

```
bool T_LCD_conf_struct::display_en
```

Screen display enable flag

Definition at line 135 of file LCD.h.

3.18.2.5 `entryModeDir`

```
bool T_LCD_conf_struct::entryModeDir
```

Entry mode direction configuration

Definition at line 138 of file LCD.h.

3.18.2.6 entryModeShift

```
bool T_LCD_conf_struct::entryModeShift
```

Entry mode shift configuration

Definition at line 139 of file LCD.h.

3.18.2.7 fontType_cnf

```
bool T_LCD_conf_struct::fontType_cnf
```

Font configuration

Definition at line 134 of file LCD.h.

3.18.2.8 i2c_addr

```
uint8_t T_LCD_conf_struct::i2c_addr
```

I²C address if the screen

Definition at line 131 of file LCD.h.

3.18.2.9 i2c_bitrate

```
uint32_t T_LCD_conf_struct::i2c_bitrate
```

I²C bitrate needed by the LCD screen

Definition at line 130 of file LCD.h.

3.18.2.10 lineNumber_cnf

```
bool T_LCD_conf_struct::lineNumber_cnf
```

Screen line number configuration (1 or 2 lines)

Definition at line 133 of file LCD.h.

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

- [LCD.h](#)

3.19 LinkedList::T_LL_element Struct Reference

Type defining a linked list element.

Collaboration diagram for LinkedList::T_LL_element:



Public Attributes

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

3.19.1 Detailed Description

Type defining a linked list element.

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

Definition at line 117 of file `LinkedList.h`.

3.19.2 Member Data Documentation

3.19.2.1 `data_ptr`

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

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

3.19.2.2 `nextElement`

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

Definition at line 120 of file `LinkedList.h`.

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

- [LinkedList.h](#)

3.20 scheduler::Task_t Struct Reference

Type defining a task structure.

Public Attributes

- `TaskPtr_t TaskPtr`
- `uint16_t period`

3.20.1 Detailed Description

Type defining a task structure.

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

Definition at line 129 of file scheduler.h.

3.20.2 Member Data Documentation

3.20.2.1 period

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

Period of the task

Definition at line 132 of file scheduler.h.

3.20.2.2 TaskPtr

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

Pointer to the task

Definition at line 131 of file scheduler.h.

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

- `scheduler.h`

3.21 TempSensor Class Reference

Class for temperature sensor.

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

Public Member Functions

- **TempSensor ()**
Class constructor.
- **uint16_t * getTempPtr ()**
Get pointer to data raw_temperature.
- **uint16_t * getHumPtr ()**
Get pointer to data raw_humidity.
- **bool getTemp (uint16_t *temp)**
Get temperature data.
- **bool getHumidity (uint16_t *hum)**
Get humidity data.
- **void setValidity (bool validity)**
Set data val_validity.
- **void updateLastValidValues ()**
- **uint8_t GetTempInteger ()**
Temperature formatting function - Integer part.
- **uint8_t GetTempDecimal ()**
Temperature formatting function - Decimal part.
- **uint8_t GetHumInteger ()**
Humidity formatting function - Integer part.
- **uint8_t GetHumDecimal ()**
Humidity formatting function - Decimal part.
- **bool GetValidity ()**
Data validity get function.
- **bool updateTaskPeriod (uint16_t period)**
Task period update.
- **uint16_t getTaskPeriod ()**
Task period get function.

Static Public Member Functions

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

Private Attributes

- **uint16_t read_temperature**
- **uint16_t read_humidity**
- **bool validity_last_read**
- **bool validity**
- **uint32_t valid_pit**
- **uint16_t valid_temp**
- **uint16_t valid_hum**
- **uint16_t task_period**

3.21.1 Detailed Description

Class for temperature sensor.

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

Definition at line 21 of file TempSensor.h.

3.21.2 Constructor & Destructor Documentation

3.21.2.1 TempSensor()

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

Class constructor.

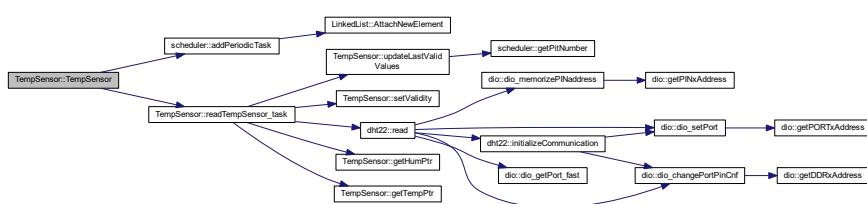
This function initializes all data of the class `TempSensor`. If needed, it creates a new instance of the DHT22 sensor object. It also adds the periodic task in the scheduler.

Returns

Nothing

Definition at line 24 of file `TempSensor.cpp`.

Here is the call graph for this function:



3.21.3 Member Function Documentation

3.21.3.1 GetHumDecimal()

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

Humidity formatting function - Decimal part.

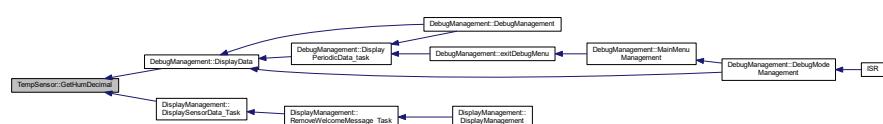
This function return the decimal part of the humidity

Returns

Decimal value of the humidity

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

Here is the caller graph for this function:



3.21.3.2 getHumidity()

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

Get humidity data.

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

Parameters

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

Returns

Validity of humidity

Definition at line 80 of file TempSensor.cpp.

3.21.3.3 GetHumInteger()

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

Humidity formatting function - Integer part.

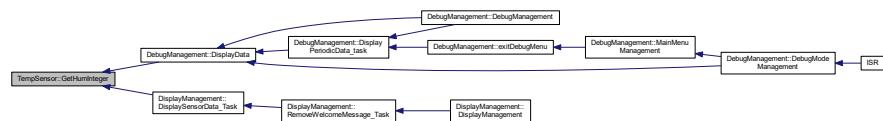
This function return the integer part of the humidity

Returns

Integer value of the humidity

Definition at line 113 of file TempSensor.h.

Here is the caller graph for this function:



3.21.3.4 getHumPtr()

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

Get pointer to data raw_humidity.

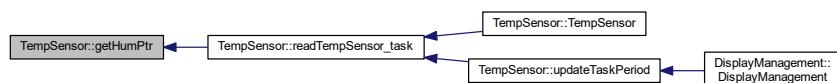
This function returns a pointer to the class member raw_humidity

Returns

Pointer to raw_humidity

Definition at line 55 of file TempSensor.cpp.

Here is the caller graph for this function:



3.21.3.5 getTaskPeriod()

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

Task period get function.

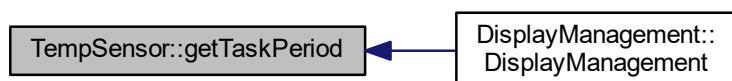
This function returns the period of the sensor task

Returns

Period of the task (ms)

Definition at line 155 of file TempSensor.h.

Here is the caller graph for this function:



3.21.3.6 getTemp()

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

Get temperature data.

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

Parameters

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

Returns

Validity of temperature

Definition at line 86 of file TempSensor.cpp.

3.21.3.7 GetTempDecimal()

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

Temperature formatting function - Decimal part.

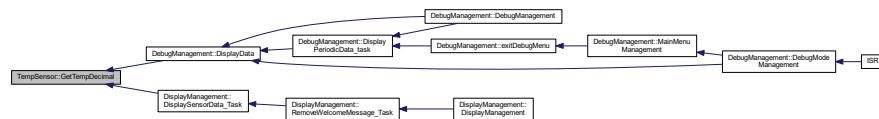
This function return the decimal part of the temperature

Returns

Decimal value of the temperature

Definition at line 102 of file TempSensor.h.

Here is the caller graph for this function:

**3.21.3.8 GetTempInteger()**

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

Temperature formatting function - Integer part.

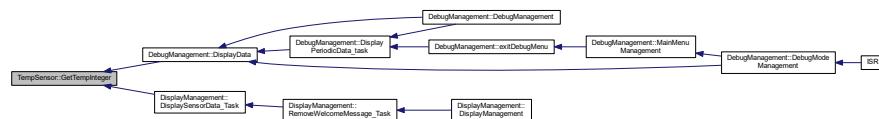
This function return the integer part of the temperature

Returns

Integer value of the temperature

Definition at line 91 of file TempSensor.h.

Here is the caller graph for this function:



3.21.3.9 getTempPtr()

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

Get pointer to data raw_temperature.

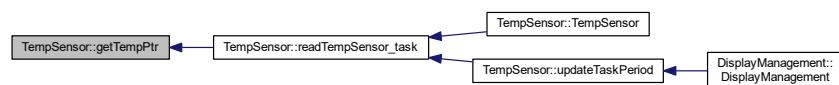
This function returns a pointer to the class member raw_temperature

Returns

Pointer to raw_temperature

Definition at line 60 of file TempSensor.cpp.

Here is the caller graph for this function:



3.21.3.10 GetValidity()

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

Data validity get function.

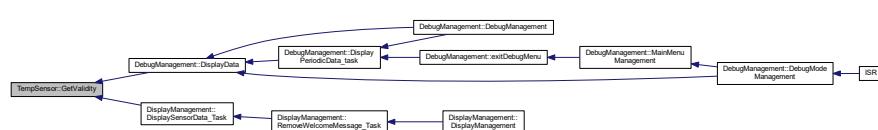
This function returns the validity of the sensor data

Returns

True if the sensor values are valid, false otherwise

Definition at line 135 of file TempSensor.h.

Here is the caller graph for this function:



3.21.3.11 readTempSensor_task()

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

Task for reading temperature and humidity values.

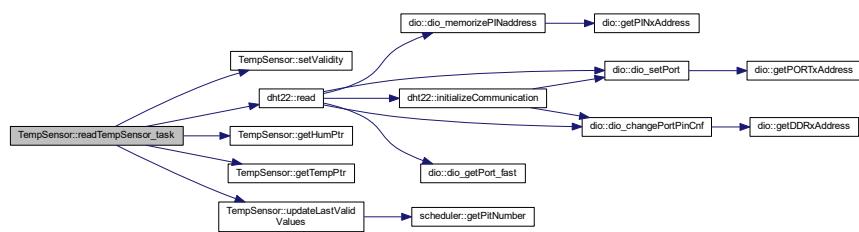
This task reads temperature and humidity data using DHT22 driver. It is called every 5 seconds.

Returns

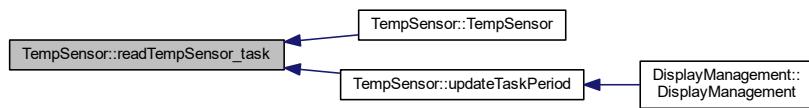
Nothing

Definition at line 44 of file TempSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.21.3.12 setValidity()

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

Set data val_validity.

This function sets the class member val_validity

Parameters

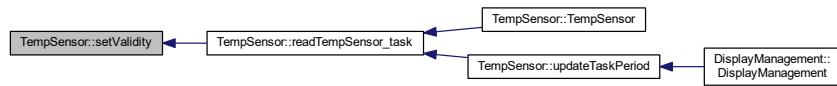
in	validity	Value of validity
----	----------	-------------------

Returns

Nothing

Definition at line 50 of file TempSensor.cpp.

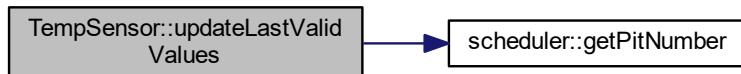
Here is the caller graph for this function:

**3.21.3.13 updateLastValidValues()**

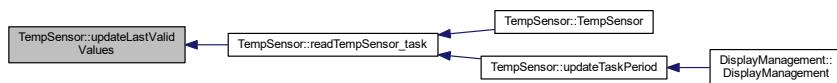
```
void TempSensor::updateLastValidValues ( )
```

Definition at line 65 of file TempSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.21.3.14 updateTaskPeriod()**

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

Task period update.

This function updates the period of the temperature task.

Parameters

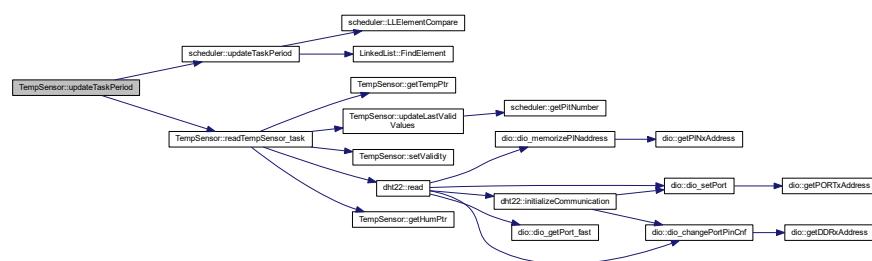
in	<i>period</i>	New period of the task
----	---------------	------------------------

Returns

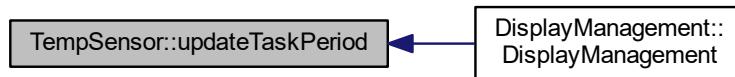
True if the period has been updated, false otherwise

Definition at line 92 of file TempSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.21.4 Member Data Documentation****3.21.4.1 read_humidity**

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

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

Definition at line 163 of file TempSensor.h.

3.21.4.2 `read_temperature`

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

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

Definition at line 162 of file TempSensor.h.

3.21.4.3 `task_period`

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

Task period

Definition at line 172 of file TempSensor.h.

3.21.4.4 `valid_hum`

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

Valid value of humidity

Definition at line 170 of file TempSensor.h.

3.21.4.5 `valid坑`

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

pit number of the last time when data were valid

Definition at line 167 of file TempSensor.h.

3.21.4.6 `valid_temp`

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

Valid value of temperature

Definition at line 169 of file TempSensor.h.

3.21.4.7 validity

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

validity of official temperature and humidity data

Definition at line 166 of file TempSensor.h.

3.21.4.8 validity_last_read

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

Validity of last read temperature and humidity data

Definition at line 164 of file TempSensor.h.

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

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

3.22 timer Class Reference

Class defining a timer.

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

Public Member Functions

- [timer \(\)](#)
Class constructor.
- void [configureTimer1](#) (uint16_t a_prescaler, uint16_t a_ctcValue)
Configures Timer #1.
- void [startTimer1 \(\)](#)
Start Timer #1.
- void [stopTimer1 \(\)](#)
Stops Timer #1.
- uint16_t [getTimer1Value \(\)](#)
Reads current value of timer #1.

Private Attributes

- uint8_t [prescaler](#)

3.22.1 Detailed Description

Class defining a timer.

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

Definition at line 19 of file timer.h.

3.22.2 Constructor & Destructor Documentation

3.22.2.1 timer()

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

Class constructor.

This function initializes class attributes

Returns

Nothing

Definition at line 15 of file timer.cpp.

3.22.3 Member Function Documentation

3.22.3.1 configureTimer1()

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

Configures Timer #1.

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

Parameters

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

Returns

Nothing

Definition at line 20 of file timer.cpp.

Here is the caller graph for this function:



3.22.3.2 getTimer1Value()

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

Reads current value of timer #1.

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

Returns

Current timer value

Definition at line 58 of file timer.h.

Here is the caller graph for this function:



3.22.3.3 startTimer1()

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

Start Timer #1.

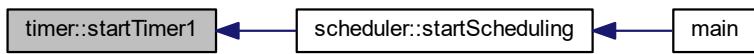
This functions starts Timer #1. Timer shall be initialized before this function is called.

Returns

Nothing

Definition at line 58 of file timer.cpp.

Here is the caller graph for this function:



3.22.3.4 stopTimer1()

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

Stops Timer #1.

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

Returns

Nothing

Definition at line 69 of file timer.cpp.

3.22.4 Member Data Documentation

3.22.4.1 prescaler

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

Definition at line 64 of file timer.h.

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

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

3.23 usart Class Reference

USART serial bus class.

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

Public Member Functions

- `uart (uint16_t a_BaudRate)`
Class usart constructor.
- `void usart_sendString (String *str)`
Send a string on USART link.
- `void usart_sendByte (uint8_t data)`
Send a single byte on USART link.
- `void setBaudRate (uint16_t a_BaudRate)`
Setting baud rate.
- `void usart_init ()`
USART hardware initialization.
- `uint8_t usart_read ()`
USART read function.

Private Member Functions

- `void usart_transmit (uint8_t Data)`
USART Transmit data.

Private Attributes

- `uint16_t BaudRate`

3.23.1 Detailed Description

USART serial bus class.

This class defines all useful functions for USART serial bus

Definition at line 16 of file usart.h.

3.23.2 Constructor & Destructor Documentation

3.23.2.1 usart()

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

Class usart constructor.

Initializes the class and call hardware initialization function

Parameters

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

Returns

Nothing.

Definition at line 18 of file usart.cpp.

Here is the call graph for this function:



3.23.3 Member Function Documentation

3.23.3.1 setBaudRate()

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

Setting baud rate.

This function sets the attribute BaudRate of the class usart

Parameters

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

Returns

Nothing

Definition at line 74 of file usart.cpp.

3.23.3.2 usart_init()

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

USART hardware initialization.

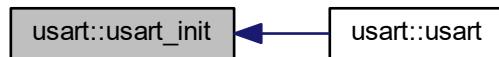
This function will initialize the USART using selected baudrate. User must pay attention to select one of the usually used Baud Rate (9600, 19200, 38400, 57600). Note that since an uint16 is used as argument, Baud rate cannot be more than 57600.

Returns

Nothing.

Definition at line 25 of file usart.cpp.

Here is the caller graph for this function:



3.23.3.3 usart_read()

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

USART read function.

This function will read reception register of USART

Returns

The function returns the 8 bits read from reception buffer

Definition at line 90 of file usart.cpp.

3.23.3.4 usart_sendByte()

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

Send a single byte on USART link.

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

Parameters

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

Returns

Nothing.

Definition at line 68 of file `uart.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.23.3.5 usart_sendString()**

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

Send a string on USART link.

This function writes the string object data to the serial link using `uart_transmit` function

Parameters

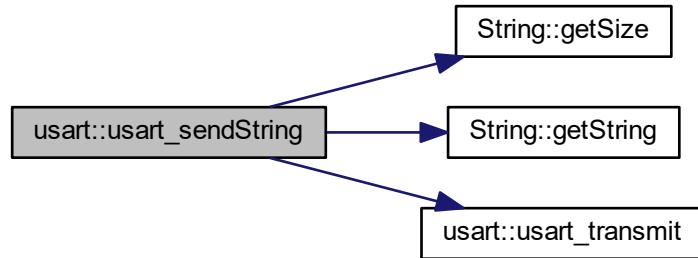
in	<i>str</i>	Pointer to the string being sent
----	------------	----------------------------------

Returns

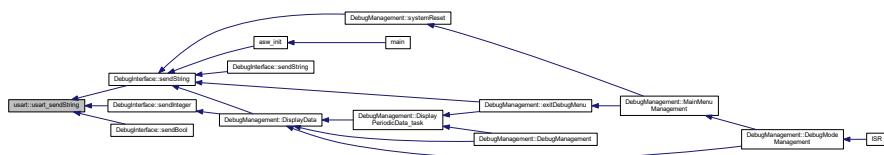
Nothing.

Definition at line 48 of file `uart.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



3.23.3.6 usart_transmit()

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

USART Transmit data.

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

Parameters

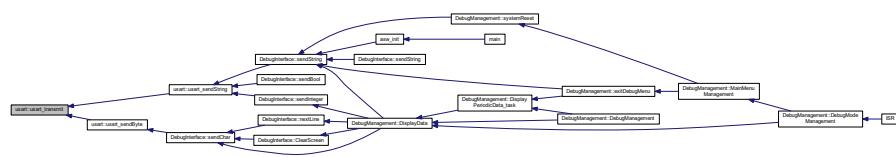
in	<i>Data</i>	Desired data char to transmit
----	-------------	-------------------------------

Returns

Nothing.

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

Here is the caller graph for this function:



3.23.4 Member Data Documentation

3.23.4.1 BaudRate

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

Defines the baud rate used by driver

Definition at line 77 of file usart.h.

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

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

3.24 Watchdog Class Reference

Watchdog management class.

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

Public Member Functions

- **Watchdog** ()
Class constructor.
 - **Watchdog** (uint8_t timeout)
Overloaded class constructor.
 - void **reset** ()
Watchdog reset function.
 - void **timeoutUpdate** (uint8_t value)
Watchdog timeout value update function.
 - void **SystemReset** ()
System reset function.
 - uint16_t **getTMOValue** ()
Watchdog timeout get value.
 - bool **isEnabled** ()
Watchdog status function.
 - bool **SwitchWdg** ()
Watchdog switching function.
 - **T_WDG_Reset_type getResetReason** ()
Reset reason get function.

Private Member Functions

- void `disable ()`
Watchdog disabling function.
- void `enable (uint8_t value)`
Watchdog enabling function.
- void `extractResetReason ()`
Reset reason reading function.

Private Attributes

- `uint8_t tmo_value`
- `bool isActive`
- `T_WDG_Reset_type reset_reason`

3.24.1 Detailed Description

`Watchdog` management class.

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

Definition at line 46 of file Watchdog.h.

3.24.2 Constructor & Destructor Documentation

3.24.2.1 Watchdog() [1/2]

`Watchdog::Watchdog ()`

Class constructor.

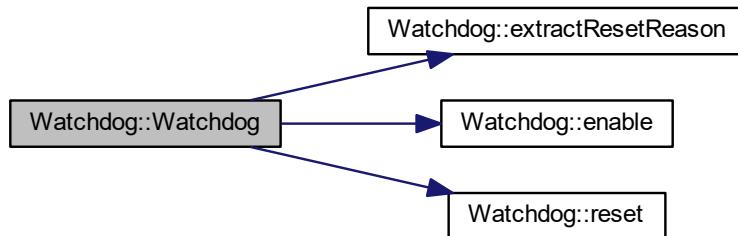
This function initializes the watchdog class. It enables the HW watchdog with a default timeout value.

Returns

Nothing

Definition at line 43 of file Watchdog.cpp.

Here is the call graph for this function:



3.24.2.2 Watchdog() [2/2]

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

Overloaded class constructor.

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

Parameters

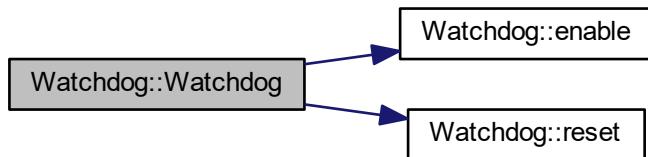
in	<i>timeout</i>	Timeout value requested for the watchdog
----	----------------	--

Returns

Nothing

Definition at line 54 of file Watchdog.cpp.

Here is the call graph for this function:



3.24.3 Member Function Documentation

3.24.3.1 disable()

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

[Watchdog](#) disabling function.

This function disables the watchdog by calling wdt_disable macro.

Returns

Nothing

Definition at line 72 of file Watchdog.cpp.

Here is the caller graph for this function:

**3.24.3.2 enable()**

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

[Watchdog](#) enabling function.

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

Parameters

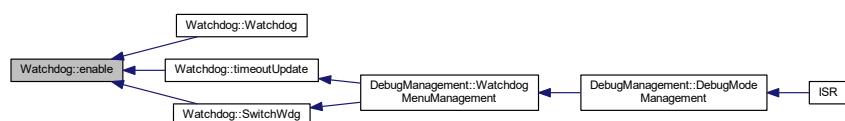
in	value	Timeout value
----	-------	---------------

Returns

Nothing

Definition at line 62 of file Watchdog.cpp.

Here is the caller graph for this function:



3.24.3.3 extractResetReason()

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

Reset reason reading function.

This function reads the reason of the last reset in register MCUSR and clears the register. The reset reason is stored in variable `reset_reason`.

Returns

Nothing

Definition at line 155 of file Watchdog.cpp.

Here is the caller graph for this function:



3.24.3.4 getResetReason()

```
T_WDG_Reset_Type Watchdog::getResetReason ( ) [inline]
```

Reset reason get function.

This function returns the reason of the last reset.

Returns

Reason of the last reset.

Definition at line 126 of file Watchdog.h.

3.24.3.5 getTMOValue()

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

[Watchdog](#) timeout get value.

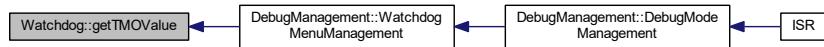
This function returns the current watchdog timeout value in ms. It has to convert the value of tmo_value into a numeric value of the timeout.

Returns

Timeout value.

Definition at line 103 of file Watchdog.cpp.

Here is the caller graph for this function:



3.24.3.6 isEnabled()

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

[Watchdog](#) status function.

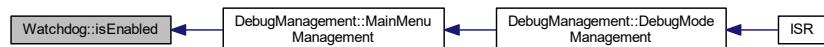
This function returns the current status of the watchdog : enabled or disabled.

Returns

True if the watchdog is enabled, false otherwise.

Definition at line 106 of file Watchdog.h.

Here is the caller graph for this function:



3.24.3.7 reset()

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

[Watchdog](#) reset function.

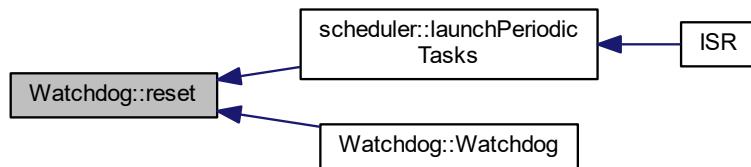
This function resets the watchdog timer by calling wdt_reset macro

Returns

Nothing

Definition at line 81 of file Watchdog.cpp.

Here is the caller graph for this function:



3.24.3.8 SwitchWdg()

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

[Watchdog](#) switching function.

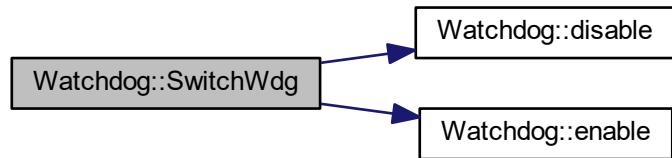
This function switches the state of the watchdog. If it was enabled, the function disables the watchdog, and if it was disabled, the function enables it with the memorized timeout value. The function returns the new status of the watchdog.

Returns

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

Definition at line 145 of file Watchdog.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**3.24.3.9 SystemReset()**

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

System reset function.

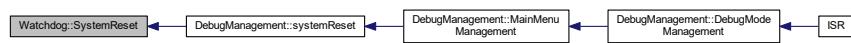
This function provokes a system reset by going in an infinite loop. Thus the watchdog will reset the CPU when the timeout occurs.

Returns

Nothing

Definition at line 98 of file Watchdog.cpp.

Here is the caller graph for this function:



3.24.3.10 timeoutUpdate()

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

[Watchdog](#) timeout value update function.

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

Parameters

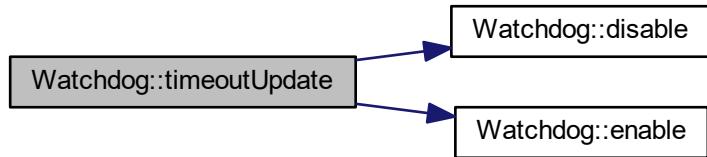
in	value	New timeout value
----	-------	-------------------

Returns

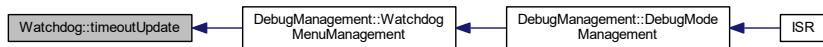
Nothing

Definition at line 86 of file Watchdog.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.24.4 Member Data Documentation

3.24.4.1 isActive

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

[Watchdog](#) activation flag

Definition at line 134 of file Watchdog.h.

3.24.4.2 reset_reason

```
T_WDG_Reset_type Watchdog::reset_reason [private]
```

Variable used to store the reason if the last reset

Definition at line 135 of file Watchdog.h.

3.24.4.3 tmo_value

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

Current timeout value

Definition at line 133 of file Watchdog.h.

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

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

Chapter 4

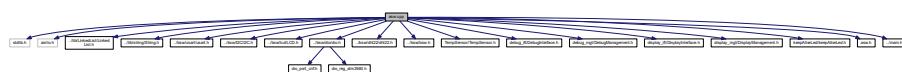
File Documentation

4.1 asw.cpp File Reference

ASW main file.

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

Include dependency graph for asw.cpp:



Functions

- void **asw_init ()**

Initialization of ASW.

4.1.1 Detailed Description

ASW main file.

Date

15 mars 2018

Author

nicls67

4.1.2 Function Documentation

4.1.2.1 asw_init()

```
void asw_init ( )
```

Initialization of ASW.

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

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

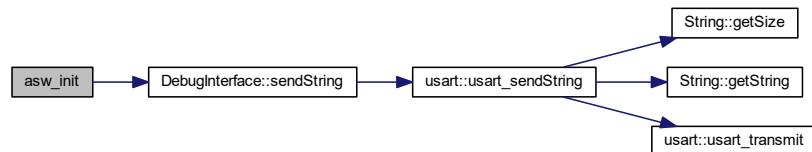
This function shall be called after BSW initialization function.

Returns

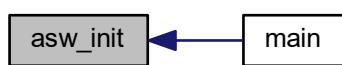
Nothing

Definition at line 37 of file asw.cpp.

Here is the call graph for this function:



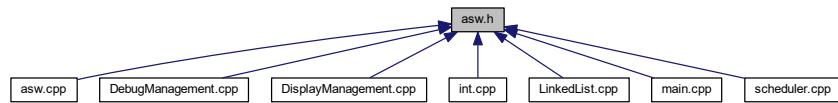
Here is the caller graph for this function:



4.2 asw.h File Reference

ASW main header file.

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



Classes

- struct [T_ASW_init_cnf](#)
ASW initialization configuration structure.

Functions

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

4.2.1 Detailed Description

ASW main header file.

Date

15 mars 2018

Author

nicls67

4.2.2 Function Documentation

4.2.2.1 asw_init()

```
void asw_init ( )
```

Initialization of ASW.

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

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

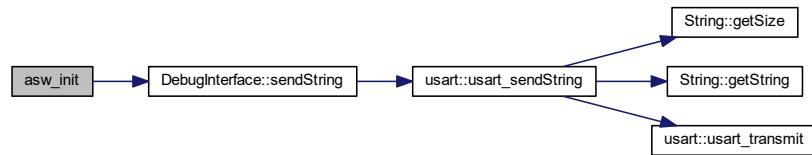
This function shall be called after BSW initialization function.

Returns

Nothing

Definition at line 37 of file asw.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



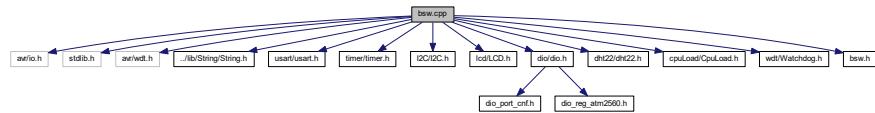
4.3 bsw.cpp File Reference

BSW main file.

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

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

Include dependency graph for bsw.cpp:
```



Functions

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

Initialization of BSW.

4.3.1 Detailed Description

BSW main file.

Date

13 mars 2018

Author

nicls67

4.3.2 Function Documentation

4.3.2.1 bsw_init()

```
void bsw_init ( )
```

Initialization of BSW.

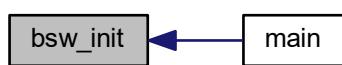
This function instantiates all driver objects, leading hardware initialization. The addresses of driver objects are then stored in BSW_cnf_struct structure.

Returns

Nothing

Definition at line 26 of file bsw.cpp.

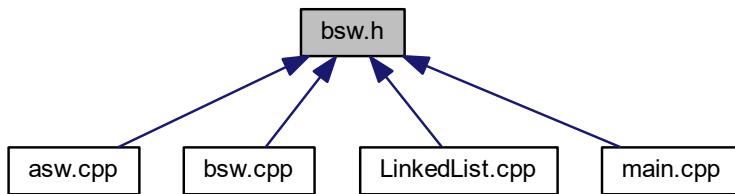
Here is the caller graph for this function:



4.4 bsw.h File Reference

BSW main header file.

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



Functions

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

Initialization of BSW.

4.4.1 Detailed Description

BSW main header file.

Date

13 mars 2018

Author

nicls67

4.4.2 Function Documentation

4.4.2.1 bsw_init()

```
void bsw_init ( )
```

Initialization of BSW.

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

Returns

Nothing

Definition at line 26 of file bsw.cpp.

Here is the caller graph for this function:

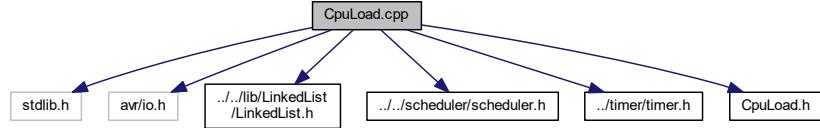


4.5 CpuLoad.cpp File Reference

Defines functions of class [CpuLoad](#).

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

Include dependency graph for CpuLoad.cpp:



Variables

- [CpuLoad * p_global_BSW_cpuload](#)

4.5.1 Detailed Description

Defines functions of class [CpuLoad](#).

Date

21 mars 2019

Author

nicls67

4.5.2 Variable Documentation

4.5.2.1 p_global_BSW_cpuload

[CpuLoad](#)* p_global_BSW_cpuload

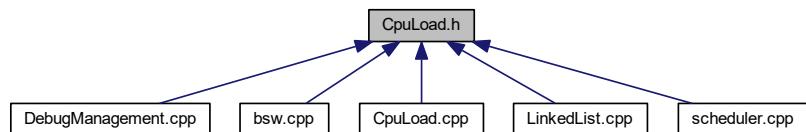
Pointer to cpu load library object

Definition at line 18 of file CpuLoad.cpp.

4.6 CpuLoad.h File Reference

[CpuLoad](#) class header file.

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



Classes

- class [CpuLoad](#)

Class defining CPU load libraries.

Macros

- #define [NB_OF_SAMPLES](#) 50

Variables

- [CpuLoad * p_global_BSW_cpuload](#)

4.6.1 Detailed Description

[CpuLoad](#) class header file.

Date

21 mars 2019

Author

nicls67

4.6.2 Macro Definition Documentation

4.6.2.1 NB_OF_SAMPLES

```
#define NB_OF_SAMPLES 50
```

Definition at line 13 of file CpuLoad.h.

4.6.3 Variable Documentation

4.6.3.1 p_global_BSW_cpuload

```
CpuLoad* p_global_BSW_cpuload
```

Pointer to cpu load library object

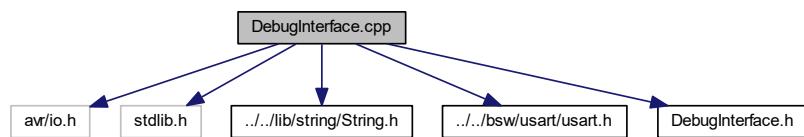
Definition at line 18 of file CpuLoad.cpp.

4.7 DebugInterface.cpp File Reference

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

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

Include dependency graph for DebugInterface.cpp:



Variables

- `DebugInterface * p_global_ASW_DebugInterface`

4.7.1 Detailed Description

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

Date

15 mars 2018

Author

nicls67

4.7.2 Variable Documentation

4.7.2.1 `p_global_ASW_DebugInterface`

`DebugInterface* p_global_ASW_DebugInterface`

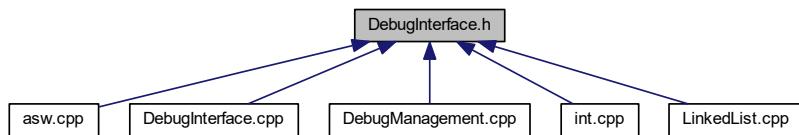
Pointer to USART debug interface object

Definition at line 19 of file DebugInterface.cpp.

4.8 DebugInterface.h File Reference

Header file for debug and logging functions.

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



Classes

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

Macros

- `#define USART_BAUDRATE (uint16_t)9600`

Variables

- `DebugInterface * p_global_ASW_DebugInterface`

4.8.1 Detailed Description

Header file for debug and logging functions.

Date

15 mars 2018

Author

nicls67

4.8.2 Macro Definition Documentation

4.8.2.1 USART_BAUDRATE

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

uart connection to PC uses a baud rate of 9600

Definition at line 15 of file DebugInterface.h.

4.8.3 Variable Documentation

4.8.3.1 p_global_ASW_DebugInterface

```
DebugInterface* p_global_ASW_DebugInterface
```

Pointer to USART debug interface object

Definition at line 19 of file DebugInterface.cpp.

4.9 DebugManagement.cpp File Reference

Debug management class source file.

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

Include dependency graph for DebugManagement.cpp:



Variables

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

4.9.1 Detailed Description

Debug management class source file.

Date

8 mai 2019

Author

nicls67

4.9.2 Variable Documentation

4.9.2.1 p_global_ASW_DebugManagement

`DebugManagement * p_global_ASW_DebugManagement`

Pointer to the `DebugManagement` object

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

4.9.2.2 str_debug_info_message_wdg_disabled

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

Info menu string displayed when the watchdog has been disabled.

Definition at line 91 of file DebugManagement.cpp.

4.9.2.3 str_debug_info_message_wdg_enabled

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

Info menu string displayed when the watchdog has been enabled.

Definition at line 96 of file DebugManagement.cpp.

4.9.2.4 str_debug_info_message_wdg_tmo_updated

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

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

Definition at line 81 of file DebugManagement.cpp.

4.9.2.5 str_debug_info_message_wdg_tmo_value

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

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

Definition at line 86 of file DebugManagement.cpp.

4.9.2.6 str_debug_info_message_wrong_menu_selection

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

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

Definition at line 76 of file DebugManagement.cpp.

4.9.2.7 str_debug_main_menu

```
const uint8_t str_debug_main_menu[ ]
```

Initial value:

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

Main menu of debug mode.

Definition at line 37 of file DebugManagement.cpp.

4.9.2.8 str_debug_wdg_menu

```
const uint8_t str_debug_wdg_menu[ ]
```

Initial value:

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

Watchdog menu of debug mode.

Definition at line 47 of file DebugManagement.cpp.

4.9.2.9 str_debug_wdg_timeout_update_selection

```
const uint8_t str_debug_wdg_timeout_update_selection[ ]
```

Initial value:

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

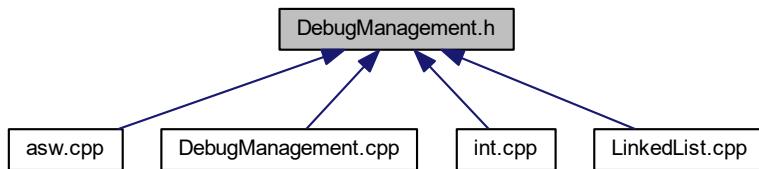
Watchdog timeout update selection.

Definition at line 58 of file DebugManagement.cpp.

4.10 DebugManagement.h File Reference

Debug management class header file.

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



Classes

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

Macros

- `#define PERIOD_MS_TASK_DISPLAY_DEBUG_DATA 5000`
- `#define PERIOD_MS_TASK_DISPLAY_CPU_LOAD 5000`

Enumerations

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

Variables

- `DebugManagement * p_global_ASW_DebugManagement`

4.10.1 Detailed Description

Debug management class header file.

Date

8 mai 2019

Author

nicls67

4.10.2 Macro Definition Documentation

4.10.2.1 PERIOD_MS_TASK_DISPLAY_CPU_LOAD

```
#define PERIOD_MS_TASK_DISPLAY_CPU_LOAD 5000
```

Period for displaying CPU load data

Definition at line 14 of file DebugManagement.h.

4.10.2.2 PERIOD_MS_TASK_DISPLAY_DEBUG_DATA

```
#define PERIOD_MS_TASK_DISPLAY_DEBUG_DATA 5000
```

Period for displaying temperature and humidity data

Definition at line 13 of file DebugManagement.h.

4.10.3 Enumeration Type Documentation

4.10.3.1 debug_mgt_main_menu_state_t

```
enum debug_mgt_main_menu_state_t
```

Defines the debug states.

Enumerator

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

Definition at line 20 of file DebugManagement.h.

4.10.3.2 debug_mgt_wdg_state_t

```
enum debug_mgt_wdg_state_t
```

Defines possible states for watchdog management.

Enumerator

WDG_MAIN	Main menu of watchdog management
WDG_TMO_UPDATE	Timeout update mode

Definition at line 30 of file DebugManagement.h.

4.10.4 Variable Documentation

4.10.4.1 p_global_ASW_DebugManagement

`DebugManagement* p_global_ASW_DebugManagement`

Pointer to the [DebugManagement](#) object

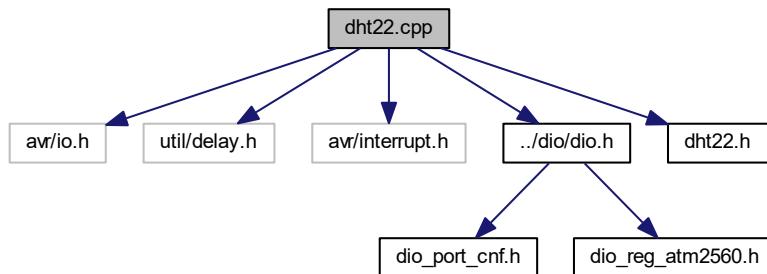
Definition at line 30 of file DebugManagement.cpp.

4.11 dht22.cpp File Reference

This file defines classes for DHT22 driver.

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "../dio/dio.h"
#include "dht22.h"
```

Include dependency graph for dht22.cpp:



Macros

- `#define MAX_WAIT_TIME_US 100`

Variables

- `dht22 * p_global_BSW_dht22`

4.11.1 Detailed Description

This file defines classes for DHT22 driver.

Date

23 mars 2018

Author

nicls67

4.11.2 Macro Definition Documentation

4.11.2.1 MAX_WAIT_TIME_US

```
#define MAX_WAIT_TIME_US 100
```

Maximum waiting time in microseconds

Definition at line 18 of file dht22.cpp.

4.11.3 Variable Documentation

4.11.3.1 p_global_BSW_dht22

```
dht22* p_global_BSW_dht22
```

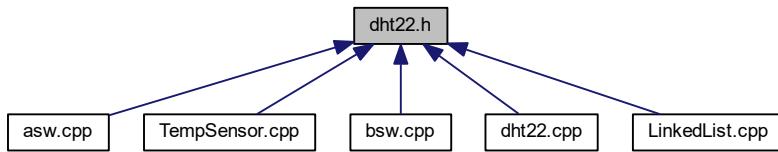
Pointer to `dht22` driver object

Definition at line 20 of file dht22.cpp.

4.12 dht22.h File Reference

DHT22 driver header file.

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



Classes

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

Variables

- [dht22 * p_global_BSW_dht22](#)

4.12.1 Detailed Description

DHT22 driver header file.

Date

23 mars 2018

Author

nicls67

4.12.2 Variable Documentation

4.12.2.1 [p_global_BSW_dht22](#)

[dht22*](#) [p_global_BSW_dht22](#)

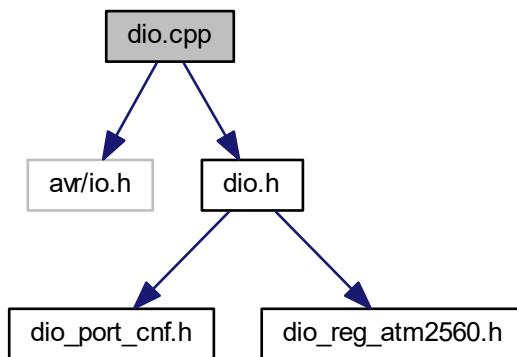
Pointer to [dht22](#) driver object

Definition at line 20 of file [dht22.cpp](#).

4.13 dio.cpp File Reference

DIO library.

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



Variables

- [dio * p_global_BSW_dio](#)

4.13.1 Detailed Description

DIO library.

Date

13 mars 2018

Author

nicls67

4.13.2 Variable Documentation

4.13.2.1 p_global_BSW_dio

`dio* p_global_BSW_dio`

Pointer to dio driver object

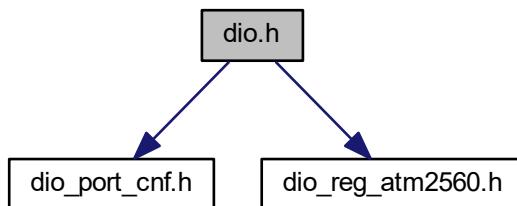
Definition at line 14 of file dio.cpp.

4.14 dio.h File Reference

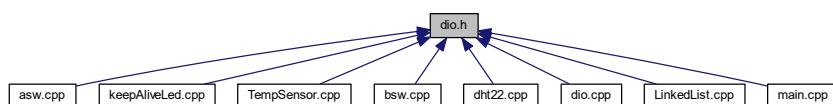
DIO library header file.

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

Include dependency graph for dio.h:



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



Classes

- class `dio`
DIO class.

Macros

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

Variables

- `dio * p_global_BSW_dio`

4.14.1 Detailed Description

DIO library header file.

Date

13 mars 2018

Author

nicls67

4.14.2 Macro Definition Documentation

4.14.2.1 DECODE_PIN

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

Macro used to extract pin index

Definition at line 20 of file dio.h.

4.14.2.2 DECODE_PORT

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

Macro used to extract port index

Definition at line 19 of file dio.h.

4.14.2.3 ENCODE_PORT

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

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

Definition at line 18 of file dio.h.

4.14.2.4 PORT_CNF_IN

```
#define PORT_CNF_IN 0
```

Pin is configured as input

Definition at line 16 of file dio.h.

4.14.2.5 PORT_CNF_OUT

```
#define PORT_CNF_OUT 1
```

Pin is configured as output

Definition at line 15 of file dio.h.

4.14.3 Variable Documentation

4.14.3.1 p_global_BSW_dio

```
dio* p_global_BSW_dio
```

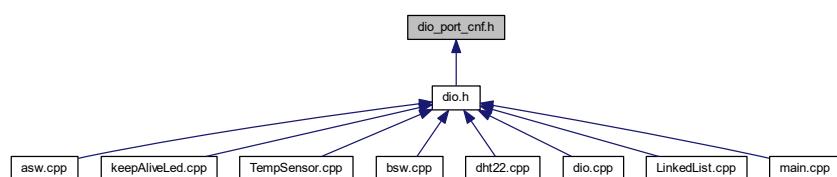
Pointer to dio driver object

Definition at line 14 of file dio.cpp.

4.15 dio_port_cnf.h File Reference

Digital ports configuration file.

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



Macros

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

4.15.1 Detailed Description

Digital ports configuration file.

Date

19 mars 2019

Author

nicls67

4.15.2 Macro Definition Documentation

4.15.2.1 PORT_A

`#define PORT_A 0`

PORTA index

Definition at line 42 of file dio_port_cnf.h.

4.15.2.2 PORT_B

`#define PORT_B 1`

PORTB index

Definition at line 43 of file dio_port_cnf.h.

4.15.2.3 PORT_C

```
#define PORT_C 2
```

PORTC index

Definition at line 44 of file dio_port_cnf.h.

4.15.2.4 PORT_D

```
#define PORT_D 3
```

PORTD index

Definition at line 45 of file dio_port_cnf.h.

4.15.2.5 PORTB_CNF_DDRC

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

Defines the configuration of DDRC register.

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

PB0 : N/A

PB1 : N/A

PB2 : N/A

PB3 : N/A

PB4 : IN

PB5 : N/A

PB6 : OUT

PB7 : OUT

Definition at line 25 of file dio_port_cnf.h.

4.15.2.6 PORTB_CNF_PORTB

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

Defines the configuration of PORTB register.

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

PB0 : N/A

PB1 : N/A

PB2 : N/A

PB3 : N/A

PB4 : Pull-up

PB5 : N/A

PB6 : HIGH

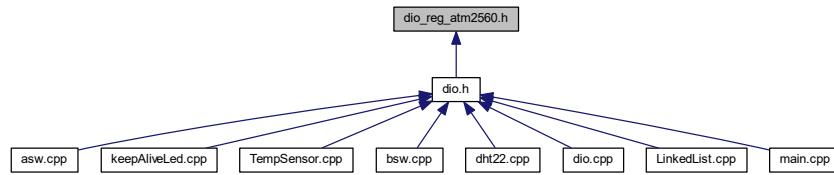
PB7 : LOW

Definition at line 40 of file dio_port_cnf.h.

4.16 dio_reg_atm2560.h File Reference

Defines DIO register addresses for ATMEGA2560.

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



Macros

- #define PORTA_PTR (volatile uint8_t *)(0x02 + 0x20)
- #define PORTB_PTR (volatile uint8_t *)(0x05 + 0x20)
- #define PORTC_PTR (volatile uint8_t *)(0x08 + 0x20)
- #define PORTD_PTR (volatile uint8_t *)(0x0B + 0x20)
- #define PINA_PTR (volatile uint8_t *)(0x00 + 0x20)
- #define PINB_PTR (volatile uint8_t *)(0x03 + 0x20)
- #define PINC_PTR (volatile uint8_t *)(0x06 + 0x20)
- #define PIND_PTR (volatile uint8_t *)(0x09 + 0x20)
- #define DDRA_PTR (volatile uint8_t *)(0x01 + 0x20)
- #define DDRB_PTR (volatile uint8_t *)(0x04 + 0x20)
- #define DDRC_PTR (volatile uint8_t *)(0x07 + 0x20)
- #define DDRD_PTR (volatile uint8_t *)(0x0A + 0x20)

4.16.1 Detailed Description

Defines DIO register addresses for ATMEGA2560.

Date

19 mars 2019

Author

nicls67

4.16.2 Macro Definition Documentation

4.16.2.1 DDRA_PTR

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

Macro defining pointer to DDR A register

Definition at line 24 of file dio_reg_atm2560.h.

4.16.2.2 DDRB_PTR

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

Macro defining pointer to DDR B register

Definition at line 25 of file dio_reg_atm2560.h.

4.16.2.3 DDRC_PTR

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

Macro defining pointer to DDR C register

Definition at line 26 of file dio_reg_atm2560.h.

4.16.2.4 DDRD_PTR

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

Macro defining pointer to DDR D register

Definition at line 27 of file dio_reg_atm2560.h.

4.16.2.5 PINA_PTR

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

Macro defining pointer to PIN A register

Definition at line 19 of file dio_reg_atm2560.h.

4.16.2.6 PINB_PTR

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

Macro defining pointer to PIN B register

Definition at line 20 of file dio_reg_atm2560.h.

4.16.2.7 PINC_PTR

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

Macro defining pointer to PIN C register

Definition at line 21 of file dio_reg_atm2560.h.

4.16.2.8 PIND_PTR

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

Macro defining pointer to PIN D register

Definition at line 22 of file dio_reg_atm2560.h.

4.16.2.9 PORTA_PTR

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

Macro defining pointer to PORT A register

Definition at line 14 of file dio_reg_atm2560.h.

4.16.2.10 PORTB_PTR

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

Macro defining pointer to PORT B register

Definition at line 15 of file dio_reg_atm2560.h.

4.16.2.11 PORTC_PTR

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

Macro defining pointer to PORT C register

Definition at line 16 of file dio_reg_atm2560.h.

4.16.2.12 PORTD_PTR

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

Macro defining pointer to PORT D register

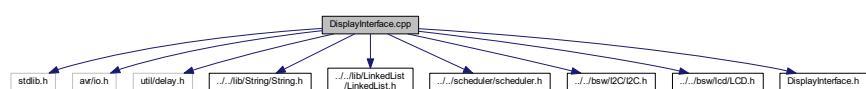
Definition at line 17 of file dio_reg_atm2560.h.

4.17 DisplayInterface.cpp File Reference

Source code file for display services.

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

Include dependency graph for DisplayInterface.cpp:



Variables

- [DisplayInterface * p_global_ASW_DisplayInterface](#)

4.17.1 Detailed Description

Source code file for display services.

Date

23 avr. 2019

Author

nicls67

4.17.2 Variable Documentation

4.17.2.1 p_global_ASW_DisplayInterface

`DisplayInterface* p_global_ASW_DisplayInterface`

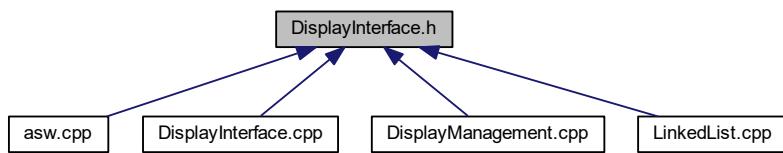
Pointer to `DisplayInterface` object

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

4.18 DisplayInterface.h File Reference

`DisplayInterface` class header file.

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



Classes

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

Macros

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

Enumerations

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

Variables

- `DisplayInterface * p_global_ASW_DisplayInterface`

4.18.1 Detailed Description

`DisplayInterface` class header file.

Date

23 avr. 2019

Author

nicls67

4.18.2 Macro Definition Documentation

4.18.2.1 DISPLAY_LINE_SHIFT_PERIOD_MS

```
#define DISPLAY_LINE_SHIFT_PERIOD_MS 500
```

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

Definition at line 68 of file `DisplayInterface.h`.

4.18.2.2 DISPLAY_LINE_SHIFT_TEMPO_TIME

```
#define DISPLAY_LINE_SHIFT_TEMPO_TIME 6
```

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

Definition at line 69 of file `DisplayInterface.h`.

4.18.3 Enumeration Type Documentation

4.18.3.1 T_DisplayInterface_LineAlignment

```
enum T_DisplayInterface_LineAlignment
```

Alignment mode for line display.

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

Enumerator

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

Definition at line 33 of file DisplayInterface.h.

4.18.3.2 T_DisplayInterface_LineDisplayMode

```
enum T_DisplayInterface_LineDisplayMode
```

Modes for line display.

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

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

Enumerator

NORMAL	
LINE_SHIFT	
GO_TO_NEXT_LINE	

Definition at line 20 of file DisplayInterface.h.

4.18.4 Variable Documentation

4.18.4.1 p_global_ASW_DisplayInterface

```
DisplayInterface* p_global_ASW_DisplayInterface
```

Pointer to [DisplayInterface](#) object

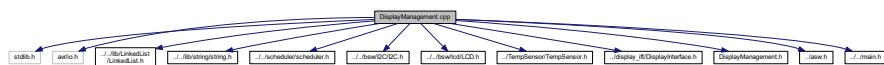
Definition at line 25 of file DisplayInterface.cpp.

4.19 DisplayManagement.cpp File Reference

Display management source file.

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

Include dependency graph for DisplayManagement.cpp:



Variables

- [DisplayManagement * p_global_ASW_DisplayManagement](#)

4.19.1 Detailed Description

Display management source file.

Date

1 mai 2019

Author

nicls67

4.19.2 Variable Documentation

4.19.2.1 p_global_ASW_DisplayManagement

[DisplayManagement* p_global_ASW_DisplayManagement](#)

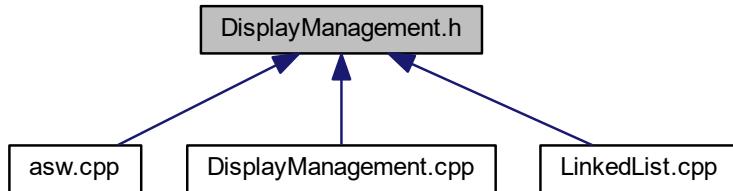
Pointer to [DisplayManagement](#) object

Definition at line 28 of file DisplayManagement.cpp.

4.20 DisplayManagement.h File Reference

Display management class header file.

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



Classes

- class [DisplayManagement](#)

Display management class.

Macros

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

Variables

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

4.20.1 Detailed Description

Display management class header file.

Date

1 mai 2019

Author

nicls67

4.20.2 Macro Definition Documentation

4.20.2.1 DISPLAY_MGT_I2C_BITRATE

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

I2C bus bitrate is 100 kHz

Definition at line 21 of file DisplayManagement.h.

4.20.2.2 DISPLAY_MGT_LCD_I2C_ADDR

```
#define DISPLAY_MGT_LCD_I2C_ADDR 0x27
```

I2C address of the screen

Definition at line 13 of file DisplayManagement.h.

4.20.2.3 DISPLAY_MGT_LINE_HUM

```
#define DISPLAY_MGT_LINE_HUM 1
```

Current humidity is displayed on line 1

Definition at line 19 of file DisplayManagement.h.

4.20.2.4 DISPLAY_MGT_LINE_TEMP

```
#define DISPLAY_MGT_LINE_TEMP 0
```

Current temperature is displayed on line 0

Definition at line 18 of file DisplayManagement.h.

4.20.2.5 DISPLAY_MGT_PERIOD_TASK_SENSOR

```
#define DISPLAY_MGT_PERIOD_TASK_SENSOR 1500
```

Display is updated every 1.5s

Definition at line 15 of file DisplayManagement.h.

4.20.2.6 DISPLAY_MGT_PERIOD_WELCOME_MSG_REMOVAL

```
#define DISPLAY_MGT_PERIOD_WELCOME_MSG_REMOVAL 5000
```

Time after which one the welcome message is removed

Definition at line 16 of file DisplayManagement.h.

4.20.3 Variable Documentation

4.20.3.1 humidityDisplayString

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

[String](#) used for humidity display

Definition at line 43 of file DisplayManagement.h.

4.20.3.2 LCD_init_cnf

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

Initial value:

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

[LCD](#) configuration structure.

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

Definition at line 27 of file DisplayManagement.h.

4.20.3.3 noSensorDisplayString

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

String used in case temperature sensor is deactivated

Definition at line 44 of file DisplayManagement.h.

4.20.3.4 p_global_ASW_DisplayManagement

```
DisplayManagement* p_global_ASW_DisplayManagement
```

Pointer to [DisplayManagement](#) object

Definition at line 28 of file DisplayManagement.cpp.

4.20.3.5 tempDisplayString

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

String used for temperature display

Definition at line 42 of file DisplayManagement.h.

4.20.3.6 welcomeMessageString

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

Definition at line 41 of file DisplayManagement.h.

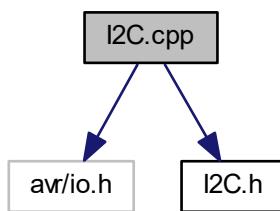
4.21 I2C.cpp File Reference

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

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

```
#include "I2C.h"
```

Include dependency graph for I2C.cpp:



Variables

- `I2C * p_global_BSW_i2c`

4.21.1 Detailed Description

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

Date

19 avr. 2019

Author

nicls67

4.21.2 Variable Documentation

4.21.2.1 `p_global_BSW_i2c`

`I2C* p_global_BSW_i2c`

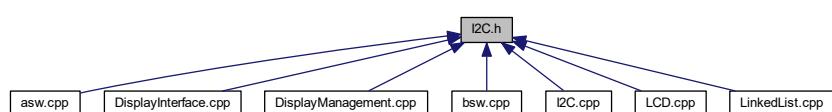
Pointer to [I2C](#) driver object

Definition at line 14 of file I2C.cpp.

4.22 I2C.h File Reference

[I2C](#) class header file.

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



Classes

- class [I2C](#)

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

Macros

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

Variables

- I2C * p_global_BSW_i2c

4.22.1 Detailed Description

I2C class header file.

Date

19 avr. 2019

Author

nicls67

4.22.2 Macro Definition Documentation

4.22.2.1 DATA_ACK

```
#define DATA_ACK 0x28
```

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

Definition at line 15 of file I2C.h.

4.22.2.2 SLA_ACK

```
#define SLA_ACK 0x18
```

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

Definition at line 14 of file I2C.h.

4.22.2.3 START

```
#define START 0x08
```

TWSR status code : START condition transmitted

Definition at line 13 of file I2C.h.

4.22.3 Variable Documentation

4.22.3.1 p_global_BSW_i2c

```
I2C* p_global_BSW_i2c
```

Pointer to [I2C](#) driver object

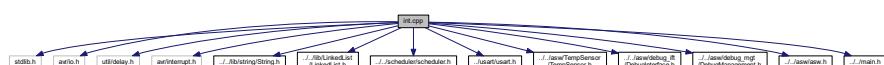
Definition at line 14 of file I2C.cpp.

4.23 int.cpp File Reference

Interrupt management source file.

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

Include dependency graph for int.cpp:



Functions

- [ISR \(TIMER1_COMPA_vect\)](#)

Main software interrupt.

- [ISR \(USART0_RX_vect\)](#)

USART Rx Complete interrupt.

4.23.1 Detailed Description

Interrupt management source file.

Date

22 mai 2019

Author

nicls67

4.23.2 Function Documentation

4.23.2.1 ISR() [1/2]

```
ISR (
    TIMER1_COMPA_vect )
```

Main software interrupt.

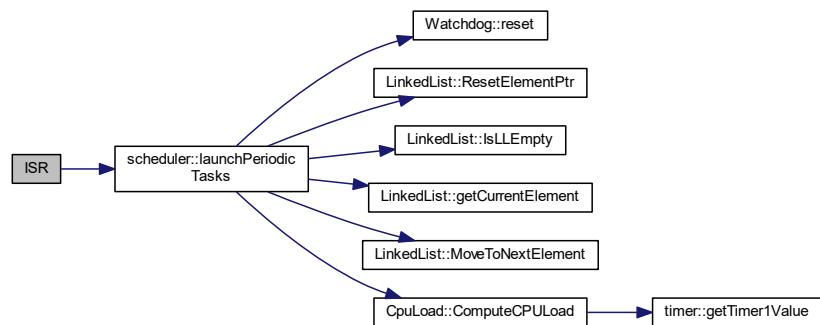
This function handles the interrupt raised by Timer #1. It wakes up the software every 500 ms to perform applications.

Returns

Nothing

Definition at line 34 of file int.cpp.

Here is the call graph for this function:



4.23.2.2 ISR() [2 / 2]

```
ISR (
    USART0_RX_vect
)
```

USART Rx Complete interrupt.

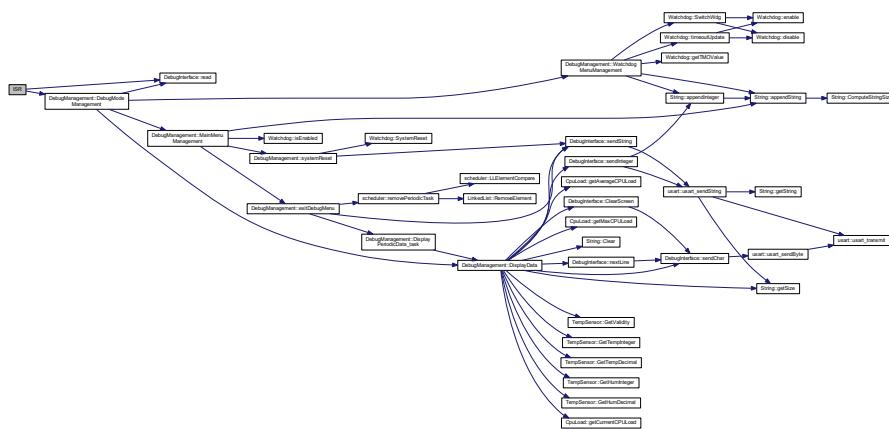
This function handles the interrupt raised when a frame has been received by USART. If debug mode mode is active, it calls debug mode management function. If inactive, it calls debug mode activation function if the received character is 'a'

Returns

Nothing

Definition at line 46 of file int.cpp.

Here is the call graph for this function:

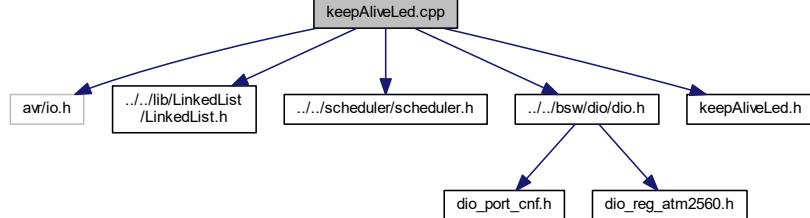


4.24 keepAliveLed.cpp File Reference

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

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

Include dependency graph for keepAliveLed.cpp:



Variables

- `keepAliveLed * p_global_ASW_keepAliveLed`

4.24.1 Detailed Description

Definition of function for class `keepAliveLed`.

Date

17 mars 2018

Author

nicls67

4.24.2 Variable Documentation

4.24.2.1 `p_global_ASW_keepAliveLed`

`keepAliveLed* p_global_ASW_keepAliveLed`

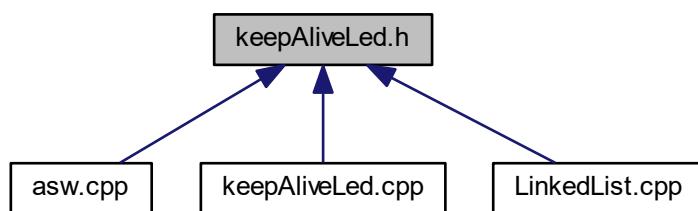
Pointer to `keepAliveLed` object

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

4.25 `keepAliveLed.h` File Reference

Class `keepAliveLed` header file.

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



Classes

- class [keepAliveLed](#)
Class for keep-alive LED blinking.

Macros

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

Variables

- [keepAliveLed * p_global_ASW_keepAliveLed](#)

4.25.1 Detailed Description

Class [keepAliveLed](#) header file.

Date

17 mars 2018

Author

nicls67

4.25.2 Macro Definition Documentation

4.25.2.1 LED_PORT

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

LED is connected to port PB7

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

4.25.2.2 PERIOD_MS_TASK_LED

```
#define PERIOD_MS_TASK_LED SW_PERIOD_MS
```

Period for led blinking

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

4.25.3 Variable Documentation

4.25.3.1 p_global_ASW_keepAliveLed

`keepAliveLed* p_global_ASW_keepAliveLed`

Pointer to [keepAliveLed](#) object

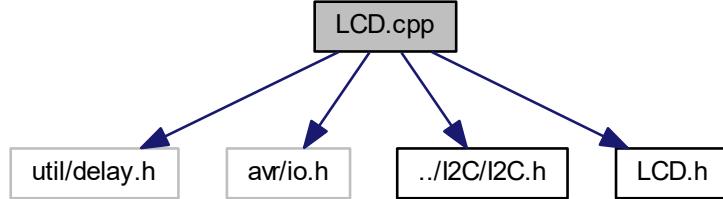
Definition at line 20 of file `keepAliveLed.cpp`.

4.26 LCD.cpp File Reference

[LCD](#) class source file.

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

Include dependency graph for `LCD.cpp`:



Variables

- `LCD * p_global_BSW_lcd`

4.26.1 Detailed Description

[LCD](#) class source file.

Date

20 avr. 2019

Author

nicls67

4.26.2 Variable Documentation

4.26.2.1 p_global_BSW_lcd

`LCD* p_global_BSW_lcd`

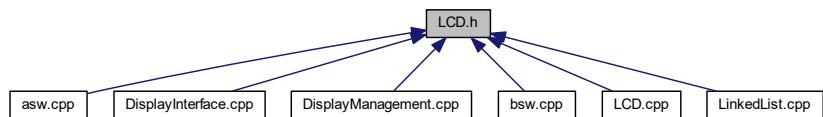
Pointer to `LCD` driver object

Definition at line 16 of file LCD.cpp.

4.27 LCD.h File Reference

`LCD` class header file.

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



Classes

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

Macros

- `#define EN_PIN 2`
- `#define RW_PIN 1`
- `#define RS_PIN 0`
- `#define BACKLIGHT_PIN 3`
- `#define LCD_INST_CLR_DISPLAY_BIT 0`
- `#define LCD_INST_FUNCTION_SET 5`
- `#define LCD_INST_DISPLAY_CTRL 3`
- `#define LCD_INST_ENTRY_MODE_SET 2`
- `#define LCD_INST_SET_DDRAM_ADDR 7`
- `#define LCD_FCT_SET_FIELD_DL 4`
- `#define LCD_FCT_SET_FIELD_N 3`
- `#define LCD_FCT_SET_FIELD_F 2`
- `#define LCD_DISPLAY_CTRL_FIELD_D 2`

- `#define LCD_DISPLAY_CTRL_FIELD_C 1`
- `#define LCD_DISPLAY_CTRL_FIELD_B 0`
- `#define LCD_CNF_SHIFT_ID 1`
- `#define LCD_CNF_SHIFT_SH 0`
- `#define LCD_CNF_ONE_LINE 0`
- `#define LCD_CNF_TWO_LINE 1`
- `#define LCD_CNF_FONT_5_8 0`
- `#define LCD_CNF_FONT_5_11 1`
- `#define LCD_CNF_DISPLAY_ON 1`
- `#define LCD_CNF_DISPLAY_OFF 0`
- `#define LCD_CNF_CURSOR_ON 1`
- `#define LCD_CNF_CURSOR_OFF 0`
- `#define LCD_CNF_CURSOR_BLINK_ON 1`
- `#define LCD_CNF_CURSOR_BLINK_OFF 0`
- `#define LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT 1`
- `#define LCD_CNF_ENTRY_MODE_DIRECTION_LEFT 0`
- `#define LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON 1`
- `#define LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF 0`
- `#define LCD_CNF_BACKLIGHT_ON 1`
- `#define LCD_CNF_BACKLIGHT_OFF 0`
- `#define LCD_RAM_1_LINE_MIN 0`
- `#define LCD_RAM_1_LINE_MAX 0x4F`
- `#define LCD_RAM_2_LINES_MIN_1 0`
- `#define LCD_RAM_2_LINES_MAX_1 0x27`
- `#define LCD_RAM_2_LINES_MIN_2 0x40`
- `#define LCD_RAM_2_LINES_MAX_2 0x67`
- `#define LCD_WAIT_CLR_RETURN 1600`
- `#define LCD_WAIT_OTHER_MODES 40`
- `#define LCD_SIZE_NB_CHAR_PER_LINE 20`
- `#define LCD_SIZE_NB_LINES 4`

Enumerations

- enum `T_LCD_command` {
 `LCD_CMD_FUNCTION_SET, LCD_CMD_CLEAR_DISPLAY, LCD_CMD_DISPLAY_CTRL, LCD_CMD_ENTRY_MODE_SET,`
`LCD_CMD_SET_DDRAM_ADDR }`

LCD commands enumeration.
- enum `T_LCD_config_mode` { `LCD_MODE_INSTRUCTION = 0, LCD_MODE_DATA = 1` }

LCD modes enumeration.
- enum `T_LCD_ram_area` { `LCD_DATA_DDRAM, LCD_DATA_CGRAM` }

Screen RAM definition.

Variables

- `LCD * p_global_BSW_lcd`

4.27.1 Detailed Description

[LCD](#) class header file.

Date

20 avr. 2019

Author

nicls67

4.27.2 Macro Definition Documentation

4.27.2.1 BACKLIGHT_PIN

```
#define BACKLIGHT_PIN 3
```

Backlight pin is on P3

Definition at line 17 of file LCD.h.

4.27.2.2 EN_PIN

```
#define EN_PIN 2
```

EN bit is on P2

Definition at line 14 of file LCD.h.

4.27.2.3 LCD_CNF_BACKLIGHT_OFF

```
#define LCD_CNF_BACKLIGHT_OFF 0
```

Backlight is disabled

Definition at line 70 of file LCD.h.

4.27.2.4 LCD_CNF_BACKLIGHT_ON

```
#define LCD_CNF_BACKLIGHT_ON 1
```

Backlight is enabled

Definition at line 69 of file LCD.h.

4.27.2.5 LCD_CNF_CURSOR_BLINK_OFF

```
#define LCD_CNF_CURSOR_BLINK_OFF 0
```

Cursor blinking is off, bit is set to 0

Definition at line 58 of file LCD.h.

4.27.2.6 LCD_CNF_CURSOR_BLINK_ON

```
#define LCD_CNF_CURSOR_BLINK_ON 1
```

Cursor blinking is on, bit is set to 1

Definition at line 57 of file LCD.h.

4.27.2.7 LCD_CNF_CURSOR_OFF

```
#define LCD_CNF_CURSOR_OFF 0
```

Cursor is off, bit is set to 0

Definition at line 54 of file LCD.h.

4.27.2.8 LCD_CNF_CURSOR_ON

```
#define LCD_CNF_CURSOR_ON 1
```

Cursor is on, bit is set to 1

Definition at line 53 of file LCD.h.

4.27.2.9 LCD_CNF_DISPLAY_OFF

```
#define LCD_CNF_DISPLAY_OFF 0
```

Display is off, bit is set to 0

Definition at line 50 of file LCD.h.

4.27.2.10 LCD_CNF_DISPLAY_ON

```
#define LCD_CNF_DISPLAY_ON 1
```

Display is on, bit is set to 1

Definition at line 49 of file LCD.h.

4.27.2.11 LCD_CNF_ENTRY_MODE_DIRECTION_LEFT

```
#define LCD_CNF_ENTRY_MODE_DIRECTION_LEFT 0
```

Direction of shift is left, bit is set to 0

Definition at line 62 of file LCD.h.

4.27.2.12 LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT

```
#define LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT 1
```

Direction of shift is right, bit is set to 1

Definition at line 61 of file LCD.h.

4.27.2.13 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF

```
#define LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF 0
```

Display shift is not performed, bit is set to 0

Definition at line 66 of file LCD.h.

4.27.2.14 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON

```
#define LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON 1
```

Display shift is performed, bit is set to 1

Definition at line 65 of file LCD.h.

4.27.2.15 LCD_CNF_FONT_5_11

```
#define LCD_CNF_FONT_5_11 1
```

Two-line configuration, bit is set to 1

Definition at line 46 of file LCD.h.

4.27.2.16 LCD_CNF_FONT_5_8

```
#define LCD_CNF_FONT_5_8 0
```

One-line configuration, bit is set to 0

Definition at line 45 of file LCD.h.

4.27.2.17 LCD_CNF_ONE_LINE

```
#define LCD_CNF_ONE_LINE 0
```

One-line configuration, bit is set to 0

Definition at line 41 of file LCD.h.

4.27.2.18 LCD_CNF_SHIFT_ID

```
#define LCD_CNF_SHIFT_ID 1
```

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

Definition at line 37 of file LCD.h.

4.27.2.19 LCD_CNF_SHIFT_SH

```
#define LCD_CNF_SHIFT_SH 0
```

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

Definition at line 38 of file LCD.h.

4.27.2.20 LCD_CNF_TWO_LINE

```
#define LCD_CNF_TWO_LINE 1
```

Two-line configuration, bit is set to 1

Definition at line 42 of file LCD.h.

4.27.2.21 LCD_DISPLAY_CTRL_FIELD_B

```
#define LCD_DISPLAY_CTRL_FIELD_B 0
```

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

Definition at line 34 of file LCD.h.

4.27.2.22 LCD_DISPLAY_CTRL_FIELD_C

```
#define LCD_DISPLAY_CTRL_FIELD_C 1
```

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

Definition at line 33 of file LCD.h.

4.27.2.23 LCD_DISPLAY_CTRL_FIELD_D

```
#define LCD_DISPLAY_CTRL_FIELD_D 2
```

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

Definition at line 32 of file LCD.h.

4.27.2.24 LCD_FCT_SET_FIELD_DL

```
#define LCD_FCT_SET_FIELD_DL 4
```

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

Definition at line 27 of file LCD.h.

4.27.2.25 LCD_FCT_SET_FIELD_F

```
#define LCD_FCT_SET_FIELD_F 2
```

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

Definition at line 29 of file LCD.h.

4.27.2.26 LCD_FCT_SET_FIELD_N

```
#define LCD_FCT_SET_FIELD_N 3
```

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

Definition at line 28 of file LCD.h.

4.27.2.27 LCD_INST_CLR_DISPLAY_BIT

```
#define LCD_INST_CLR_DISPLAY_BIT 0
```

Instruction bit for "clear display" is DB0

Definition at line 20 of file LCD.h.

4.27.2.28 LCD_INST_DISPLAY_CTRL

```
#define LCD_INST_DISPLAY_CTRL 3
```

Instruction bit for "display control" is DB3

Definition at line 22 of file LCD.h.

4.27.2.29 LCD_INST_ENTRY_MODE_SET

```
#define LCD_INST_ENTRY_MODE_SET 2
```

Instruction bit for "entry mode" is DB2

Definition at line 23 of file LCD.h.

4.27.2.30 LCD_INST_FUNCTION_SET

```
#define LCD_INST_FUNCTION_SET 5
```

Instruction bit for "function set" is DB5

Definition at line 21 of file LCD.h.

4.27.2.31 LCD_INST_SET_DDRAM_ADDR

```
#define LCD_INST_SET_DDRAM_ADDR 7
```

Instruction bit for "set DDRAM address" is DB7

Definition at line 24 of file LCD.h.

4.27.2.32 LCD_RAM_1_LINE_MAX

```
#define LCD_RAM_1_LINE_MAX 0x4F
```

Maximum address value in 1-line mode

Definition at line 74 of file LCD.h.

4.27.2.33 LCD_RAM_1_LINE_MIN

```
#define LCD_RAM_1_LINE_MIN 0
```

Minimum address value in 1-line mode

Definition at line 73 of file LCD.h.

4.27.2.34 LCD_RAM_2_LINES_MAX_1

```
#define LCD_RAM_2_LINES_MAX_1 0x27
```

Maximum address value in 2-lines mode for line 1

Definition at line 76 of file LCD.h.

4.27.2.35 LCD_RAM_2_LINES_MAX_2

```
#define LCD_RAM_2_LINES_MAX_2 0x67
```

Maximum address value in 2-lines mode for line 2

Definition at line 78 of file LCD.h.

4.27.2.36 LCD_RAM_2_LINES_MIN_1

```
#define LCD_RAM_2_LINES_MIN_1 0
```

Minimum address value in 2-lines mode for line 1

Definition at line 75 of file LCD.h.

4.27.2.37 LCD_RAM_2_LINES_MIN_2

```
#define LCD_RAM_2_LINES_MIN_2 0x40
```

Minimum address value in 2-lines mode for line 2

Definition at line 77 of file LCD.h.

4.27.2.38 LCD_SIZE_NB_CHAR_PER_LINE

```
#define LCD_SIZE_NB_CHAR_PER_LINE 20
```

LCD screen has 20 characters per line

Definition at line 85 of file LCD.h.

4.27.2.39 LCD_SIZE_NB_LINES

```
#define LCD_SIZE_NB_LINES 4
```

LCD screen has 4 lines

Definition at line 86 of file LCD.h.

4.27.2.40 LCD_WAIT_CLR_RETURN

```
#define LCD_WAIT_CLR_RETURN 1600
```

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

Definition at line 81 of file LCD.h.

4.27.2.41 LCD_WAIT_OTHER_MODES

```
#define LCD_WAIT_OTHER_MODES 40
```

Waiting time after all other modes is at least 38 us

Definition at line 82 of file LCD.h.

4.27.2.42 RS_PIN

```
#define RS_PIN 0
```

RS pin is on P0

Definition at line 16 of file LCD.h.

4.27.2.43 RW_PIN

```
#define RW_PIN 1
```

RW pin is on P1

Definition at line 15 of file LCD.h.

4.27.3 Enumeration Type Documentation

4.27.3.1 T_LCD_command

```
enum T_LCD_command
```

LCD commands enumeration.

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

Enumerator

LCD_CMDFUNCTION_SET	
LCD_CMDCLEAR_DISPLAY	
LCD_CMDDISPLAY_CTRL	
LCD_CMDENTRY_MODE_SET	
LCD_CMDSET_DDRAM_ADDR	

Definition at line 93 of file LCD.h.

4.27.3.2 T_LCD_config_mode

```
enum T_LCD_config_mode
```

LCD modes enumeration.

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

Enumerator

LCD_MODEINSTRUCTION	
LCD_MODEDATA	

Definition at line 107 of file LCD.h.

4.27.3.3 T_LCD_ram_area

```
enum T_LCD_ram_area
```

Screen RAM definition.

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

Enumerator

LCD_DATADDRAM	
LCD_DATACGRAM	

Definition at line 118 of file LCD.h.

4.27.4 Variable Documentation

4.27.4.1 p_global_BSW_lcd

`LCD* p_global_BSW_lcd`

Pointer to `LCD` driver object

Definition at line 16 of file LCD.cpp.

4.28 LinkedList.cpp File Reference

Linked List library source file.

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

Include dependency graph for LinkedList.cpp:



4.28.1 Detailed Description

Linked List library source file.

Date

27 avr. 2019

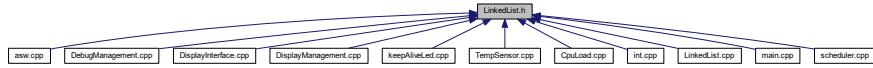
Author

nicls67

4.29 LinkedList.h File Reference

Linked List library header file.

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



Classes

- class [LinkedList](#)
Linked list class.
- struct [LinkedList::T_LL_element](#)
Type defining a linked list element.

Typedefs

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

4.29.1 Detailed Description

Linked List library header file.

Date

27 avr. 2019

Author

nicls67

4.29.2 Typedef Documentation

4.29.2.1 CompareFctPtr_t

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

Type defining a pointer to the comparison function

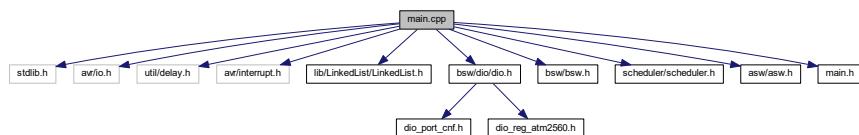
Definition at line 14 of file `LinkedList.h`.

4.30 main.cpp File Reference

Background task file.

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

Include dependency graph for main.cpp:



Macros

- #define DEBUG_ACTIVE_PORT ENCODE_PORT(PORT_B, 4)

Functions

- int main (void)

Background task of program.

Variables

- bool isDebugModeActivated
- const T_ASW_init_cnf ASW_init_cnf

4.30.1 Detailed Description

Background task file.

Date

12 mars 2018

Author

nicls67

4.30.2 Macro Definition Documentation

4.30.2.1 DEBUG_ACTIVE_PORT

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

Debug activation pin is port PB6

Definition at line 26 of file main.cpp.

4.30.3 Function Documentation

4.30.3.1 main()

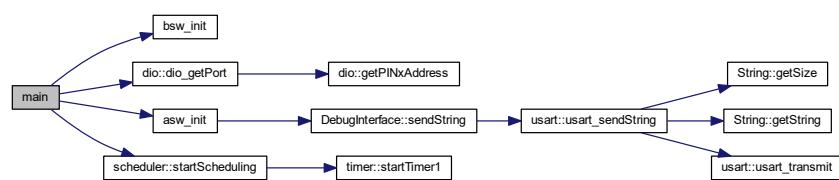
```
int main (
    void )
```

Background task of program.

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

Definition at line 45 of file main.cpp.

Here is the call graph for this function:



4.30.4 Variable Documentation

4.30.4.1 ASW_init_cnf

```
const T_ASW_init_cnf ASW_init_cnf
```

Initial value:

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

Definition of needed ASW services

Definition at line 30 of file main.cpp.

4.30.4.2 isDebugEnabledActivated

```
bool isDebugEnabledActivated
```

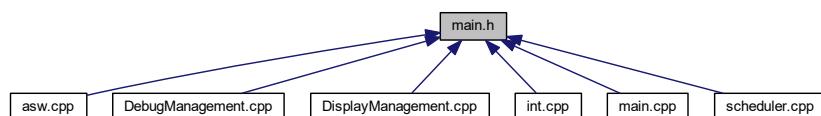
Flag indicating if the debug mode is activated or not

Definition at line 28 of file main.cpp.

4.31 main.h File Reference

Background task header file.

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



Variables

- bool `isDebugEnabledActivated`
- const `T_ASW_init_cnf ASW_init_cnf`

4.31.1 Detailed Description

Background task header file.

Date

17 mars 2018

Author

nicls67

4.31.2 Variable Documentation

4.31.2.1 ASW_init_cnf

```
const T_ASW_init_cnf ASW_init_cnf
```

Definition of needed ASW services

Definition at line 30 of file main.cpp.

4.31.2.2 isDebugEnabledActivated

```
bool isDebugEnabledActivated
```

Flag indicating if the debug mode is activated or not

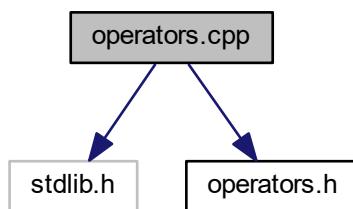
Definition at line 28 of file main.cpp.

4.32 operators.cpp File Reference

C++ operators definitions

```
#include <stdlib.h>
#include "operators.h"
```

Include dependency graph for operators.cpp:



Functions

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

4.32.1 Detailed Description

c++ operators definitions

Date

14 mars 2018

Author

nicls67

4.32.2 Function Documentation

4.32.2.1 operator delete()

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

Operator delete.

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

Parameters

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

Returns

Nothing

Definition at line 18 of file operators.cpp.

4.32.2.2 operator new()

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

Operator new.

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

Parameters

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

Returns

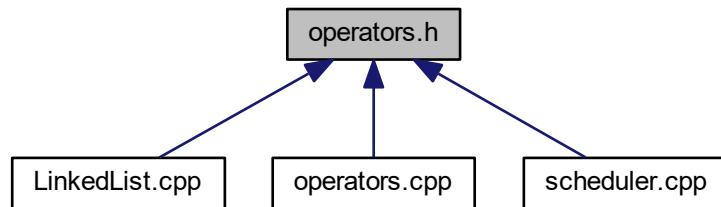
Pointer to the start of allocated memory zone

Definition at line 13 of file operators.cpp.

4.33 operators.h File Reference

c++ operators definitions header file

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



Functions

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

4.33.1 Detailed Description

c++ operators definitions header file

Date

14 mars 2018

Author

nicls67

4.33.2 Function Documentation

4.33.2.1 operator delete()

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

Operator delete.

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

Parameters

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

Returns

Nothing

Definition at line 18 of file operators.cpp.

4.33.2.2 operator new()

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

Operator new.

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

Parameters

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

Returns

Pointer to the start of allocated memory zone

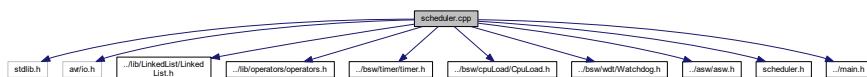
Definition at line 13 of file operators.cpp.

4.34 scheduler.cpp File Reference

Defines scheduler class.

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

Include dependency graph for scheduler.cpp:



Variables

- `scheduler * p_global_scheduler`

4.34.1 Detailed Description

Defines scheduler class.

Date

16 mars 2018

Author

nicls67

4.34.2 Variable Documentation

4.34.2.1 p_global_scheduler

`scheduler* p_global_scheduler`

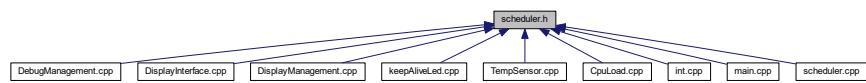
Pointer to scheduler object

Definition at line 27 of file scheduler.cpp.

4.35 scheduler.h File Reference

Scheduler class header file.

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



Classes

- class [scheduler](#)
Scheduler class.
- struct [scheduler::Task_t](#)
Type defining a task structure.

Macros

- #define [SW_PERIOD_MS](#) 500
- #define [PRESCALER_PERIODIC_TIMER](#) 256
- #define [TIMER_CTC_VALUE](#) ((F_CPU/PRESCALER_PERIODIC_TIMER)/(1000/SW_PERIOD_MS))

Typedefs

- typedef void(* [TaskPtr_t](#)) (void)
Type defining a pointer to function.

Variables

- [scheduler * p_global_scheduler](#)

4.35.1 Detailed Description

Scheduler class header file.

Date

16 mars 2018

Author

nicls67

4.35.2 Macro Definition Documentation

4.35.2.1 PRESCALER_PERIODIC_TIMER

```
#define PRESCALER_PERIODIC_TIMER 256
```

Value of prescaler to use for periodic timer

Definition at line 16 of file scheduler.h.

4.35.2.2 SW_PERIOD_MS

```
#define SW_PERIOD_MS 500
```

Software period, used to define periodic timer interrupt

Definition at line 15 of file scheduler.h.

4.35.2.3 TIMER_CTC_VALUE

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

Compare value for periodic timer

Definition at line 17 of file scheduler.h.

4.35.3 Typedef Documentation

4.35.3.1 TaskPtr_t

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

Type defining a pointer to function.

Definition at line 22 of file scheduler.h.

4.35.4 Variable Documentation

4.35.4.1 p_global_scheduler

`scheduler* p_global_scheduler`

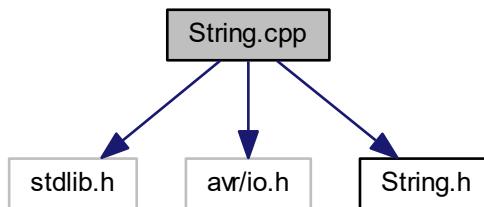
Pointer to scheduler object

Definition at line 27 of file scheduler.cpp.

4.36 String.cpp File Reference

[String](#) class source file.

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



4.36.1 Detailed Description

[String](#) class source file.

Date

2 mai 2019

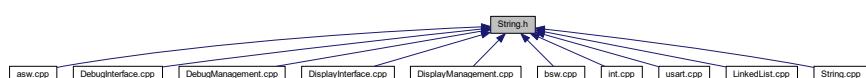
Author

nicls67

4.37 String.h File Reference

[String](#) class header file.

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



Classes

- class [String](#)

String management class.

4.37.1 Detailed Description

[String](#) class header file.

Date

2 mai 2019

Author

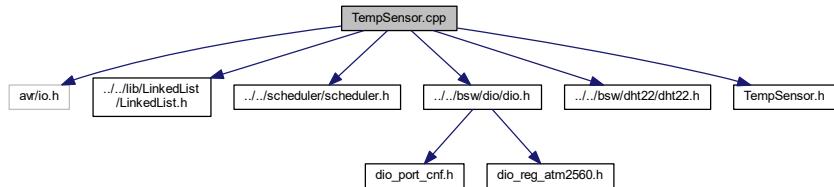
nicls67

4.38 TempSensor.cpp File Reference

Defines function of class [TempSensor](#).

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

Include dependency graph for TempSensor.cpp:



Macros

- `#define PIT_BEFORE_INVALID 60`

Variables

- `TempSensor * p_global_ASW_TempSensor`

4.38.1 Detailed Description

Defines function of class [TempSensor](#).

Date

23 mars 2018

Author

nicls67

4.38.2 Macro Definition Documentation

4.38.2.1 PIT_BEFORE_INVALID

```
#define PIT_BEFORE_INVALID 60
```

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

4.38.3 Variable Documentation

4.38.3.1 p_global_ASW_TempSensor

```
TempSensor* p_global_ASW_TempSensor
```

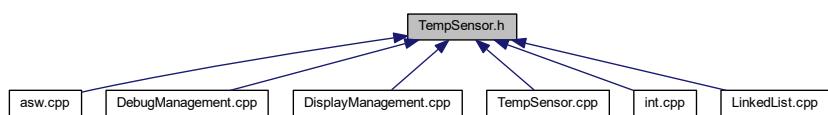
Pointer to [TempSensor](#) object

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

4.39 TempSensor.h File Reference

Class [TempSensor](#) header file.

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



Classes

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

Macros

- `#define PERIOD_MS_TASK_TEMP_SENSOR 5000`
- `#define DHT22_PORT ENCODE_PORT(PORT_B, 6)`

Variables

- `TempSensor * p_global_ASW_TempSensor`

4.39.1 Detailed Description

Class [TempSensor](#) header file.

Date

23 mars 2018

Author

nicls67

4.39.2 Macro Definition Documentation

4.39.2.1 DHT22_PORT

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

DHT22 is connected to port PB6

Definition at line 15 of file TempSensor.h.

4.39.2.2 PERIOD_MS_TASK_TEMP_SENSOR

```
#define PERIOD_MS_TASK_TEMP_SENSOR 5000
```

Period for reading temperature data

Definition at line 13 of file TempSensor.h.

4.39.3 Variable Documentation

4.39.3.1 p_global_ASW_TempSensor

`TempSensor* p_global_ASW_TempSensor`

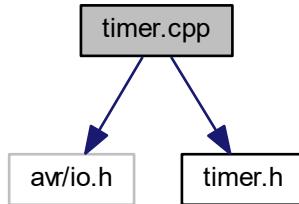
Pointer to `TempSensor` object

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

4.40 timer.cpp File Reference

Defines function for class `timer`.

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



Variables

- `timer * p_global_BSW_timer`

4.40.1 Detailed Description

Defines function for class `timer`.

Date

15 mars 2018

Author

nicls67

4.40.2 Variable Documentation

4.40.2.1 p_global_BSW_timer

`timer* p_global_BSW_timer`

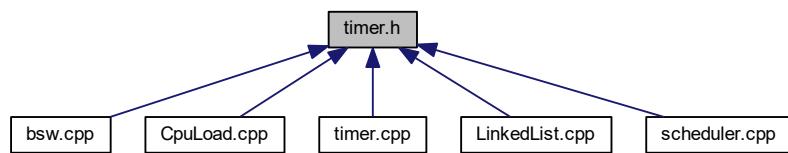
Pointer to timer driver object

Definition at line 13 of file timer.cpp.

4.41 timer.h File Reference

Timer class header file.

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



Classes

- class `timer`
Class defining a timer.

Variables

- `timer * p_global_BSW_timer`

4.41.1 Detailed Description

Timer class header file.

Date

15 mars 2018

Author

nicls67

4.41.2 Variable Documentation

4.41.2.1 p_global_BSW_timer

`timer* p_global_BSW_timer`

Pointer to timer driver object

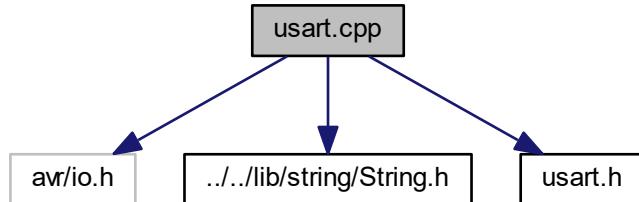
Definition at line 13 of file timer.cpp.

4.42 usart.cpp File Reference

BSW library for USART.

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

Include dependency graph for usart.cpp:



Variables

- `uart * p_global_BSW_usart`

4.42.1 Detailed Description

BSW library for USART.

Date

13 mars 2018

Author

nicls67

4.42.2 Variable Documentation

4.42.2.1 p_global_BSW_usart

`usart* p_global_BSW_usart`

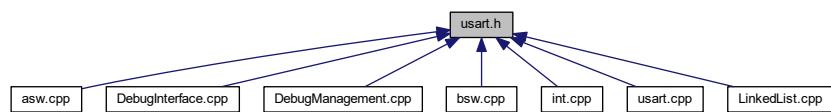
Pointer to usart driver object

Definition at line 16 of file usart.cpp.

4.43 usart.h File Reference

Header file for USART library.

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



Classes

- class `usart`
USART serial bus class.

Variables

- `usart * p_global_BSW_usart`

4.43.1 Detailed Description

Header file for USART library.

Date

13 mars 2018

Author

nicls67

4.43.2 Variable Documentation

4.43.2.1 p_global_BSW_usart

`usart* p_global_BSW_usart`

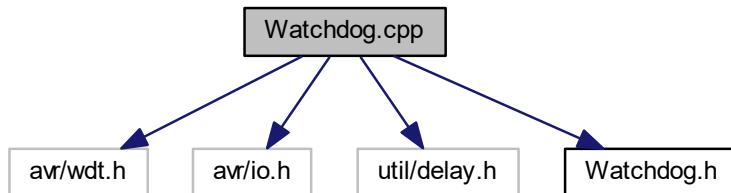
Pointer to usart driver object

Definition at line 16 of file usart.cpp.

4.44 Watchdog.cpp File Reference

Class [Watchdog](#) source code file.

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



Macros

- `#define WDG_TIMEOUT_DEFAULT_MS WDG_TMO_500MS`

Functions

- `uint8_t mcusr_mirror __attribute__ ((section(".noinit")))`
MCUSR register mirror.
- `void get_mcusr (void)`
MCUSR register get function.

Variables

- [Watchdog * p_global_BSW_wdg](#)

4.44.1 Detailed Description

Class [Watchdog](#) source code file.

Date

6 juin 2019

Author

nicls67

4.44.2 Macro Definition Documentation

4.44.2.1 WDG_TIMEOUT_DEFAULT_MS

```
#define WDG_TIMEOUT_DEFAULT_MS WDG_TMO_500MS
```

Default timeout value is set to 500 ms

Definition at line 18 of file Watchdog.cpp.

4.44.3 Function Documentation

4.44.3.1 __attribute__()

```
uint8_t mcusr_mirror __attribute__ (
    (section(".noinit")) )
```

MCUSR register mirror.

This variable mirrors the value of the MCUSR register. It is set during the init3 phase. It is global to be accessible from the watchdog class.

4.44.3.2 get_mcusr()

```
void get_mcusr (
    void )
```

MCUSR register get function.

This function stores the value of the MCUSR register into the mirror global variable. It also clears the register and disables the watchdog.

Returns

Nothing

Definition at line 36 of file Watchdog.cpp.

4.44.4 Variable Documentation

4.44.4.1 p_global_BSW_wdg

`Watchdog*` `p_global_BSW_wdg`

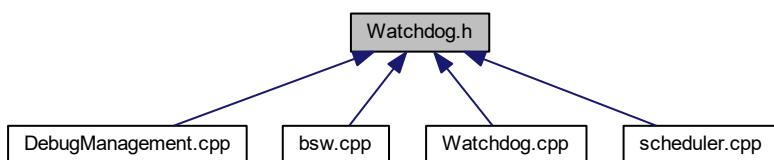
Pointer to [Watchdog](#) driver object

Definition at line 20 of file Watchdog.cpp.

4.45 Watchdog.h File Reference

Class [Watchdog](#) header file.

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



Classes

- class [Watchdog](#)
Watchdog management class.

Macros

- `#define WDG_TMO_15MS WDTO_15MS`
Definition of available timeout values.
- `#define WDG_TMO_30MS WDTO_30MS`
- `#define WDG_TMO_60MS WDTO_60MS`
- `#define WDG_TMO_120MS WDTO_120MS`
- `#define WDG_TMO_250MS WDTO_250MS`
- `#define WDG_TMO_500MS WDTO_500MS`
- `#define WDG_TMO_1S WDTO_1S`
- `#define WDG_TMO_2S WDTO_2S`
- `#define WDG_TMO_4S WDTO_4S`
- `#define WDG_TMO_8S WDTO_8S`

Enumerations

- `enum T_WDG_Reset_type {`
`JTAG_RESET_TYPE, WDG_RESET_TYPE, BROWN_OUT_RESET_TYPE, EXTERNAL_RESET_TYPE,`
`POWER_ON_RESET_TYPE }`
Type definition for reset source.

Functions

- `uint8_t mcusr_mirror __attribute__ ((section(".noinit")))`

Variables

- `Watchdog * p_global_BSW_wdg`

4.45.1 Detailed Description

Class `Watchdog` header file.

Date

6 juin 2019

Author

nicls67

4.45.2 Macro Definition Documentation

4.45.2.1 WDG_TMO_120MS

```
#define WDG_TMO_120MS WDTO_120MS
```

Timeout value is 120 ms

Definition at line 33 of file Watchdog.h.

4.45.2.2 WDG_TMO_15MS

```
#define WDG_TMO_15MS WDTO_15MS
```

Definition of available timeout values.

Timeout value is 15 ms

Definition at line 30 of file Watchdog.h.

4.45.2.3 WDG_TMO_1S

```
#define WDG_TMO_1S WDTO_1S
```

Timeout value is 1 s

Definition at line 36 of file Watchdog.h.

4.45.2.4 WDG_TMO_250MS

```
#define WDG_TMO_250MS WDTO_250MS
```

Timeout value is 250 ms

Definition at line 34 of file Watchdog.h.

4.45.2.5 WDG_TMO_2S

```
#define WDG_TMO_2S WDTO_2S
```

Timeout value is 2 s

Definition at line 37 of file Watchdog.h.

4.45.2.6 WDG_TMO_30MS

```
#define WDG_TMO_30MS WDTO_30MS
```

Timeout value is 30 ms

Definition at line 31 of file Watchdog.h.

4.45.2.7 WDG_TMO_4S

```
#define WDG_TMO_4S WDTO_4S
```

Timeout value is 4 s

Definition at line 38 of file Watchdog.h.

4.45.2.8 WDG_TMO_500MS

```
#define WDG_TMO_500MS WDTO_500MS
```

Timeout value is 500 ms

Definition at line 35 of file Watchdog.h.

4.45.2.9 WDG_TMO_60MS

```
#define WDG_TMO_60MS WDTO_60MS
```

Timeout value is 60 ms

Definition at line 32 of file Watchdog.h.

4.45.2.10 WDG_TMO_8S

```
#define WDG_TMO_8S WDTO_8S
```

Timeout value is 8 s

Definition at line 39 of file Watchdog.h.

4.45.3 Enumeration Type Documentation

4.45.3.1 T_WDG_Reset_type

```
enum T_WDG_Reset_type
```

Type definition for reset source.

This enumeration defines all possible reset sources : JTAG reset, watchdog reset, brown-out reset, external reset and power-on reset.

Enumerator

JTAG_RESET_TYPE	
WDG_RESET_TYPE	
BROWN_OUT_RESET_TYPE	
EXTERNAL_RESET_TYPE	
POWER_ON_RESET_TYPE	

Definition at line 17 of file Watchdog.h.

4.45.4 Function Documentation

4.45.4.1 __attribute__()

```
uint8_t mcusr_mirror __attribute__ (
    section(".noinit")) )
```

4.45.5 Variable Documentation

4.45.5.1 p_global_BSW_wdg

`Watchdog*` `p_global_BSW_wdg`

Pointer to [Watchdog](#) driver object

Definition at line 20 of file Watchdog.cpp.

Index

__attribute__
 Watchdog.cpp, 227
 Watchdog.h, 232
~LinkedList
 LinkedList, 85
~String
 String, 100

ASW_init_cnf
 main.cpp, 208
 main.h, 210
addPeriodicTask
 scheduler, 92
alignment
 T_display_data, 109
appendBool
 String, 101
appendChar
 String, 102
appendInteger
 String, 103
appendString
 String, 103
asw.cpp, 147
 asw_init, 148
asw.h, 149
 asw_init, 149
asw_init
 asw.cpp, 148
 asw.h, 149
AttachNewElement
 LinkedList, 86
avg_load
 CpuLoad, 8

BACKLIGHT_PIN
 LCD.h, 195
backlight_en
 T_LCD_conf_struct, 113
backlight_enable
 LCD, 80
BaudRate
 uart, 136
bitrate
 I2C, 64
blinkLed_task
 keepAliveLed, 65
bsw.cpp, 150
 bsw_init, 151
bsw.h, 152

bsw_init, 152
bsw_init
 bsw.cpp, 151
 bsw.h, 152

Clear
 String, 104
ClearFullScreen
 DisplayInterface, 45
ClearLine
 DisplayInterface, 46
ClearScreen
 DebugInterface, 12
ClearStringInDataStruct
 DisplayInterface, 47
cnfCursorBlink
 LCD, 81
cnfCursorOnOff
 LCD, 81
cnfDisplayOnOff
 LCD, 81
cnfEntryModeDir
 LCD, 81
cnfEntryModeShift
 LCD, 81
cnfFontType
 LCD, 82
cnfI2C_addr
 LCD, 82
cnfLineNumber
 LCD, 82
command
 LCD, 69
CompareFctPtr_t
 LinkedList.h, 206
ComputeCPUload
 CpuLoad, 6
ComputeStringSize
 String, 105
ConfigureBacklight
 LCD, 70
ConfigureCursorBlink
 LCD, 70
ConfigureCursorOnOff
 LCD, 71
ConfigureDisplayOnOff
 LCD, 72
ConfigureEntryModeDir
 LCD, 72
ConfigureEntryModeShift

LCD, 73
 ConfigureFontType
 LCD, 74
 ConfigureI2CAddr
 LCD, 74
 ConfigureLineNumber
 LCD, 75
 configureTimer1
 timer, 128
 CpuLoad, 5
 avg_load, 8
 ComputeCPUload, 6
 CpuLoad, 6
 current_load, 8
 getAverageCPUload, 7
 getCurrentCPUload, 7
 getMaxCPUload, 7
 last_sum_value, 8
 max_load, 9
 sample_cnt, 9
 sample_idx, 9
 sample_mem, 9
 CpuLoad.cpp, 153
 p_global_BSW_cpuload, 154
 CpuLoad.h, 154
 NB_OF_SAMPLES, 155
 p_global_BSW_cpuload, 155
 curElement_ptr
 LinkedList, 90
 current_load
 CpuLoad, 8
 cursor_en
 T_LCD_conf_struct, 113
 cursorBlink_en
 T_LCD_conf_struct, 113

 DATA_ACK
 I2C.h, 186
 DDRA_PTR
 dio_reg_atm2560.h, 173
 DDRB_PTR
 dio_reg_atm2560.h, 174
 DDRC_PTR
 dio_reg_atm2560.h, 174
 DDRD_PTR
 dio_reg_atm2560.h, 174
 DEBUG_ACTIVE_PORT
 main.cpp, 208
 DECODE_PIN
 dio.h, 169
 DECODE_PORT
 dio.h, 169
 DHT22_PORT
 TempSensor.h, 221
 DISPLAY_LINE_SHIFT_PERIOD_MS
 DisplayInterface.h, 178
 DISPLAY_LINE_SHIFT_TEMPO_TIME
 DisplayInterface.h, 178
 DISPLAY_MGT_I2C_BITRATE
 DisplayManagement.h, 182
 DISPLAY_MGT_LCD_I2C_ADDR
 DisplayManagement.h, 182
 DISPLAY_MGT_LINE_HUM
 DisplayManagement.h, 182
 DISPLAY_MGT_LINE_TEMP
 DisplayManagement.h, 182
 DISPLAY_MGT_PERIOD_TASK_SENSOR
 DisplayManagement.h, 182
 DISPLAY_MGT_PERIOD_WELCOME_MSG_REMOTE
 VAL
 DisplayManagement.h, 182
 data_ptr
 LinkedList::T_LL_element, 115
 ddram_addr
 LCD, 82
 debug_ift_ptr
 DebugManagement, 28
 debug_mgt_main_menu_state_t
 DebugManagement.h, 163
 debug_mgt_state_struct_t, 10
 main_state, 10
 wdg_state, 10
 debug_mgt_wdg_state_t
 DebugManagement.h, 163
 debug_state
 DebugManagement, 28
 DebugInterface, 11
 ClearScreen, 12
 DebugInterface, 12
 nextLine, 12
 read, 13
 sendBool, 13
 sendChar, 14
 sendInteger, 15
 sendString, 16, 17
 usart_drv_ptr, 17
 DebugInterface.cpp, 156
 p_global_ASW_DebugInterface, 156
 DebugInterface.h, 157
 p_global_ASW_DebugInterface, 158
 USART_BAUDRATE, 157
 DebugManagement, 18
 debug_ift_ptr, 28
 debug_state, 28
 DebugManagement, 19
 DebugModeManagement, 20
 DisplayData, 21
 DisplayPeriodicData_task, 22
 exitDebugMenu, 23
 getIftPtr, 24
 getInfoStringPtr, 24
 getMenuStringPtr, 25
 info_string_ptr, 28
 isInfoStringDisplayed, 29
 MainMenuManagement, 25
 menu_string_ptr, 29
 setInfoStringPtr, 26

systemReset, 26
tempSensor_ptr, 29
WatchdogMenuManagement, 27
DebugManagement.cpp, 158
 p_global_ASW_DebugManagement, 159
 str_debug_info_message_wdg_disabled, 159
 str_debug_info_message_wdg_enabled, 160
 str_debug_info_message_wdg_tmo_updated, 160
 str_debug_info_message_wdg_tmo_value, 160
 str_debug_info_message_wrong_menu_selection, 160
 str_debug_main_menu, 160
 str_debug_wdg_menu, 161
 str_debug_wdg_timeout_update_selection, 161
DebugManagement.h, 162
 debug_mgt_main_menu_state_t, 163
 debug_mgt_wdg_state_t, 163
 p_global_ASW_DebugManagement, 164
PERIOD_MS_TASK_DISPLAY_CPU_LOAD, 163
PERIOD_MS_TASK_DISPLAY_DEBUG_DATA, 163
DebugModeManagement
 DebugManagement, 20
dht22, 30
 dht22, 31
 dht22_port, 33
 dio_ptr, 33
 initializeCommunication, 31
 read, 32
dht22.cpp, 164
 MAX_WAIT_TIME_US, 165
 p_global_BSW_dht22, 165
dht22.h, 166
 p_global_BSW_dht22, 166
dht22_port
 dht22, 33
dio, 34
 dio, 35
 dio_changePortPinCnf, 35
 dio_getPort, 36
 dio_getPort_fast, 37
 dio_invertPort, 37
 dio_memorizePINaddress, 38
 dio_setPort, 39
 getDDRxAddress, 40
 getPINxAddress, 41
 getPORTxAddress, 41
 PINx_addr_mem, 42
 PINx_idx_mem, 42
 ports_init, 42
dio.cpp, 167
 p_global_BSW_dio, 167
dio.h, 168
 DECODE_PIN, 169
 DECODE_PORT, 169
 ENCODE_PORT, 169
 p_global_BSW_dio, 170
 PORT_CNF_IN, 169
PORT_CNF_OUT, 170
dio_changePortPinCnf
 dio, 35
dio_getPort
 dio, 36
dio_getPort_fast
 dio, 37
dio_invertPort
 dio, 37
dio_memorizePINaddress
 dio, 38
dio_port_cnf.h, 170
 PORT_A, 171
 PORT_B, 171
 PORT_C, 171
 PORT_D, 172
 PORTB_CNF_DDRB, 172
 PORTB_CNF_PORTB, 172
dio_ptr
 dht22, 33
dio_reg_atm2560.h, 173
 DDRA_PTR, 173
 DDRB_PTR, 174
 DDRC_PTR, 174
 DDRD_PTR, 174
 PINA_PTR, 174
 PINB_PTR, 174
 PINC_PTR, 175
 PIND_PTR, 175
 PORTA_PTR, 175
 PORTB_PTR, 175
 PORTC_PTR, 175
 PORTD_PTR, 176
dio_setPort
 dio, 39
disable
 Watchdog, 138
display_data
 DisplayInterface, 54
display_en
 T_LCD_conf_struct, 113
display_str
 T_display_data, 110
DisplayData
 DebugManagement, 21
DisplayFullLine
 DisplayInterface, 48
DisplayInterface, 43
 ClearFullScreen, 45
 ClearLine, 46
 ClearStringInDataStruct, 47
 display_data, 54
 DisplayFullLine, 48
 DisplayInterface, 45
 dummy, 54
 FindFirstCharAddr, 48
 getDisplayDataPtr, 49
 IsLineEmpty, 50

isShiftInProgress, 54
 p_lcd, 54
 RefreshLine, 50
 setLineAlignment, 51
 setLineAlignmentAndRefresh, 52
 shiftLine_task, 52
 updateLineAndRefresh, 53
DisplayInterface.cpp, 176
 p_global_ASW_DisplayInterface, 177
DisplayInterface.h, 177
 DISPLAY_LINE_SHIFT_PERIOD_MS, 178
 DISPLAY_LINE_SHIFT_TEMPO_TIME, 178
 p_global_ASW_DisplayInterface, 179
 T_DisplayInterface_LineAlignment, 178
 T_DisplayInterface_LineDisplayMode, 179
DisplayManagement, 55
 DisplayManagement, 56
 DisplaySensorData_Task, 57
 GetIftPointer, 58
 GetTempSensorPtr, 58
 p_display_ift, 60
 p_tempSensor, 60
 RemoveWelcomeMessage_Task, 59
DisplayManagement.cpp, 180
 p_global_ASW_DisplayManagement, 180
DisplayManagement.h, 181
 DISPLAY_MGT_I2C_BITRATE, 182
 DISPLAY_MGT_LCD_I2C_ADDR, 182
 DISPLAY_MGT_LINE_HUM, 182
 DISPLAY_MGT_LINE_TEMP, 182
 DISPLAY_MGT_PERIOD_TASK_SENSOR, 182
 DISPLAY_MGT_PERIOD_WELCOME_MSG_R←
 EMOVAL, 182
 humidityDisplayString, 183
 LCD_init_cnf, 183
 noSensorDisplayString, 183
 p_global_ASW_DisplayManagement, 184
 tempDisplayString, 184
 welcomeMessageString, 184
DisplayPeriodicData_task
 DebugManagement, 22
DisplaySensorData_Task
 DisplayManagement, 57
dummy
 DisplayInterface, 54
EN_PIN
 LCD.h, 195
ENCODE_PORT
 dio.h, 169
enable
 Watchdog, 139
entryModeDir
 T_LCD_conf_struct, 113
entryModeShift
 T_LCD_conf_struct, 113
exitDebugMenu
 DebugManagement, 23
extractResetReason
 Watchdog, 139
FindElement
 LinkedList, 86
FindFirstCharAddr
 DisplayInterface, 48
firstElement
 LinkedList, 90
fontType_cnf
 T_LCD_conf_struct, 114
get_mcusr
 Watchdog.cpp, 227
getAverageCPUload
 CpuLoad, 7
getCurrentElement
 LinkedList, 87
getCurrentCPUload
 CpuLoad, 7
GetDDRAMAddress
 LCD, 76
getDDRxAddress
 dio, 40
getDisplayDataPtr
 DisplayInterface, 49
GetHumDecimal
 TempSensor, 118
GetHumInteger
 TempSensor, 119
getHumPtr
 TempSensor, 119
getHumidity
 TempSensor, 118
GetIftPointer
 DisplayManagement, 58
getIftPtr
 DebugManagement, 24
getInfoStringPtr
 DebugManagement, 24
GetLineNumberCnf
 LCD, 76
getMaxCPUload
 CpuLoad, 7
getMenuStringPtr
 DebugManagement, 25
getPINxAddress
 dio, 41
getPORTxAddress
 dio, 41
getPitNumber
 scheduler, 93
getResetReason
 Watchdog, 140
getSize
 String, 105
getString
 String, 106
getTMOValue
 Watchdog, 140

getTaskCount
scheduler, 93
getTaskPeriod
TempSensor, 120
getTemp
TempSensor, 120
GetTempDecimal
TempSensor, 121
GetTempInteger
TempSensor, 121
getTempPtr
TempSensor, 121
GetTempSensorPtr
DisplayManagement, 58
getTimer1Value
timer, 129
GetValidity
TempSensor, 122
humidityDisplayString
DisplayManagement.h, 183

I2C.cpp, 184
p_global_BSW_i2c, 185
I2C.h, 185
DATA_ACK, 186
p_global_BSW_i2c, 187
SLA_ACK, 186
START, 186

I2C, 60
bitrate, 64
I2C, 61
initializeBus, 62
setBitRate, 62
setTxAddress, 63
tx_address, 64
writeByte, 63

i2c_addr
T_LCD_conf_struct, 114

i2c_bitrate
T_LCD_conf_struct, 114

i2c_drv_ptr
LCD, 82

ISR
int.cpp, 188

info_string_ptr
DebugManagement, 28

initializeBus
I2C, 62

initializeCommunication
dht22, 31

InitializeScreen
LCD, 76

int.cpp, 187
ISR, 188

isActive
Watchdog, 144

isDebugEnabled
T_ASW_init_cnf, 108

isDebugEnabled
main.cpp, 209
main.h, 210

isDisplayActivated
T_ASW_init_cnf, 108

isEmpty
T_display_data, 110

isEnabled
Watchdog, 141

isInfoStringDisplayed
DebugManagement, 29

isLEDActivated
T_ASW_init_cnf, 108

IsLLEmpty
LinkedList, 87

IsLineEmpty
DisplayInterface, 50

isShiftInProgress
DisplayInterface, 54

isTempSensorActivated
T_ASW_init_cnf, 108

keepAliveLed, 64
blinkLed_task, 65
keepAliveLed, 65

keepAliveLed.cpp, 189
p_global_ASW_keepAliveLed, 190

keepAliveLed.h, 190
LED_PORT, 191
p_global_ASW_keepAliveLed, 192
PERIOD_MS_TASK_LED, 191

LCD.cpp, 192
p_global_BSW_lcd, 193

LCD.h, 193
BACKLIGHT_PIN, 195
EN_PIN, 195
LCD_CNF_BACKLIGHT_OFF, 195
LCD_CNF_BACKLIGHT_ON, 195
LCD_CNF_CURSOR_BLINK_OFF, 196
LCD_CNF_CURSOR_BLINK_ON, 196
LCD_CNF_CURSOR_OFF, 196
LCD_CNF_CURSOR_ON, 196
LCD_CNF_DISPLAY_OFF, 196
LCD_CNF_DISPLAY_ON, 197
LCD_CNF_ENTRY_MODE_DIRECTION_LEFT, 197
LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT, 197
LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_← OFF, 197
LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_→ ON, 197
LCD_CNF_FONT_5_11, 198
LCD_CNF_FONT_5_8, 198
LCD_CNF_ONE_LINE, 198
LCD_CNF_SHIFT_ID, 198
LCD_CNF_SHIFT_SH, 198
LCD_CNF_TWO_LINE, 199

LCD_DISPLAY_CTRL_FIELD_B, 199
 LCD_DISPLAY_CTRL_FIELD_C, 199
 LCD_DISPLAY_CTRL_FIELD_D, 199
 LCD_FCT_SET_FIELD_DL, 199
 LCD_FCT_SET_FIELD_F, 200
 LCD_FCT_SET_FIELD_N, 200
 LCD_INST_CLR_DISPLAY_BIT, 200
 LCD_INST_DISPLAY_CTRL, 200
 LCD_INST_ENTRY_MODE_SET, 200
 LCD_INST_FUNCTION_SET, 201
 LCD_INST_SET_DDRAM_ADDR, 201
 LCD_RAM_1_LINE_MAX, 201
 LCD_RAM_1_LINE_MIN, 201
 LCD_RAM_2_LINES_MAX_1, 201
 LCD_RAM_2_LINES_MAX_2, 202
 LCD_RAM_2_LINES_MIN_1, 202
 LCD_RAM_2_LINES_MIN_2, 202
 LCD_SIZE_NB_CHAR_PER_LINE, 202
 LCD_SIZE_NB_LINES, 202
 LCD_WAIT_CLR_RETURN, 203
 LCD_WAIT_OTHER_MODES, 203
 p_global_BSW_lcd, 204
 RS_PIN, 203
 RW_PIN, 203
 T_LCD_command, 203
 T_LCD_config_mode, 204
 T_LCD_ram_area, 204
 LCD_CNF_BACKLIGHT_OFF
 LCD.h, 195
 LCD_CNF_BACKLIGHT_ON
 LCD.h, 195
 LCD_CNF_CURSOR_BLINK_OFF
 LCD.h, 196
 LCD_CNF_CURSOR_BLINK_ON
 LCD.h, 196
 LCD_CNF_CURSOR_OFF
 LCD.h, 196
 LCD_CNF_CURSOR_ON
 LCD.h, 196
 LCD_CNF_DISPLAY_OFF
 LCD.h, 196
 LCD_CNF_DISPLAY_ON
 LCD.h, 197
 LCD_CNF_ENTRY_MODE_DIRECTION_LEFT
 LCD.h, 197
 LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT
 LCD.h, 197
 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF
 LCD.h, 197
 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON
 LCD.h, 197
 LCD_CNF_FONT_5_11
 LCD.h, 198
 LCD_CNF_FONT_5_8
 LCD.h, 198
 LCD_CNF_ONE_LINE
 LCD.h, 198
 LCD_CNF_SHIFT_ID

LCD.h, 198
 LCD_CNF_SHIFT_SH
 LCD.h, 198
 LCD_CNF_TWO_LINE
 LCD.h, 199
 LCD_DISPLAY_CTRL_FIELD_B
 LCD.h, 199
 LCD_DISPLAY_CTRL_FIELD_C
 LCD.h, 199
 LCD_DISPLAY_CTRL_FIELD_D
 LCD.h, 199
 LCD_FCT_SET_FIELD_DL
 LCD.h, 199
 LCD_FCT_SET_FIELD_F
 LCD.h, 200
 LCD_FCT_SET_FIELD_N
 LCD.h, 200
 LCD_INST_CLR_DISPLAY_BIT
 LCD.h, 200
 LCD_INST_DISPLAY_CTRL
 LCD.h, 200
 LCD_INST_ENTRY_MODE_SET
 LCD.h, 200
 LCD_INST_FUNCTION_SET
 LCD.h, 201
 LCD_INST_SET_DDRAM_ADDR
 LCD.h, 201
 LCD_RAM_1_LINE_MAX
 LCD.h, 201
 LCD_RAM_1_LINE_MIN
 LCD.h, 201
 LCD_RAM_2_LINES_MAX_1
 LCD.h, 201
 LCD_RAM_2_LINES_MAX_2
 LCD.h, 202
 LCD_RAM_2_LINES_MIN_1
 LCD.h, 202
 LCD_RAM_2_LINES_MIN_2
 LCD.h, 202
 LCD_SIZE_NB_CHAR_PER_LINE
 LCD.h, 202
 LCD_SIZE_NB_LINES
 LCD.h, 202
 LCD_WAIT_CLR_RETURN
 LCD.h, 203
 LCD_WAIT_OTHER_MODES
 LCD.h, 203
 LCD_init_cnf
 DisplayManagement.h, 183
 LCD, 66
 backlight_enable, 80
 cnfCursorBlink, 81
 cnfCursorOnOff, 81
 cnfDisplayOnOff, 81
 cnfEntryModeDir, 81
 cnfEntryModeShift, 81
 cnfFontType, 82
 cnfI2C_addr, 82

cnfLineNumber, 82
command, 69
ConfigureBacklight, 70
ConfigureCursorBlink, 70
ConfigureCursorOnOff, 71
ConfigureDisplayOnOff, 72
ConfigureEntryModeDir, 72
ConfigureEntryModeShift, 73
ConfigureFontType, 74
ConfigureI2CAddr, 74
ConfigureLineNumber, 75
ddram_addr, 82
GetDDRAMAddress, 76
GetLineNumberCnf, 76
i2c_drv_ptr, 82
InitializeScreen, 76
LCD, 68
SetDDRAMAddress, 77
write, 78
write4bits, 79
WriteInRam, 80
LED_PORT
 keepAliveLed.h, 191
LLElementCompare
 scheduler, 94
last_sum_value
 CpuLoad, 8
launchPeriodicTasks
 scheduler, 94
lineNumber_cnf
 T_LCD_conf_struct, 114
LinkedList, 83
 ~LinkedList, 85
 AttachNewElement, 86
 curElement_ptr, 90
 FindElement, 86
 firstElement, 90
 getCurrentElement, 87
 IsLLEmpty, 87
 LinkedList, 85
 MoveToNextElement, 88
 RemoveElement, 88
 ResetElementPtr, 89
 T_LL_element, 84
LinkedList.cpp, 205
LinkedList.h, 206
 CompareFctPtr_t, 206
LinkedList::T_LL_element, 115
 data_ptr, 115
 nextElement, 115
MAX_WAIT_TIME_US
 dht22.cpp, 165
main
 main.cpp, 208
main.cpp, 207
 ASW_init_cnf, 208
 DEBUG_ACTIVE_PORT, 208
 isDebugModeActivated, 209
main, 208
main.h, 209
 ASW_init_cnf, 210
 isDebugModeActivated, 210
main_state
 debug_mgt_state_struct_t, 10
MainMenuManagement
 DebugManagement, 25
max_load
 CpuLoad, 9
menu_string_ptr
 DebugManagement, 29
mode
 T_display_data, 110
MoveToNextElement
 LinkedList, 88
NB_OF_SAMPLES
 CpuLoad.h, 155
nextElement
 LinkedList::T_LL_element, 115
nextLine
 DebugInterface, 12
noSensorDisplayString
 DisplayManagement.h, 183
operator delete
 operators.cpp, 211
 operators.h, 214
operator new
 operators.cpp, 211
 operators.h, 214
operators.cpp, 210
 operator delete, 211
 operator new, 211
operators.h, 213
 operator delete, 214
 operator new, 214
p_display_ift
 DisplayManagement, 60
p_global_ASW_DebugInterface
 DebugInterface.cpp, 156
 DebugInterface.h, 158
p_global_ASW_DebugManagement
 DebugManagement.cpp, 159
 DebugManagement.h, 164
p_global_ASW_DisplayInterface
 DisplayInterface.cpp, 177
 DisplayInterface.h, 179
p_global_ASW_DisplayManagement
 DisplayManagement.cpp, 180
 DisplayManagement.h, 184
p_global_ASW_TempSensor
 TempSensor.cpp, 220
 TempSensor.h, 222
p_global_ASW_keepAliveLed
 keepAliveLed.cpp, 190
 keepAliveLed.h, 192

p_global_BSW_cpupload
 CpuLoad.cpp, 154
 CpuLoad.h, 155
 p_global_BSW_dht22
 dht22.cpp, 165
 dht22.h, 166
 p_global_BSW_dio
 dio.cpp, 167
 dio.h, 170
 p_global_BSW_i2c
 I2C.cpp, 185
 I2C.h, 187
 p_global_BSW_lcd
 LCD.cpp, 193
 LCD.h, 204
 p_global_BSW_timer
 timer.cpp, 223
 timer.h, 224
 p_global_BSW_usart
 usart.cpp, 225
 usart.h, 226
 p_global_BSW_wdg
 Watchdog.cpp, 228
 Watchdog.h, 232
 p_global_scheduler
 scheduler.cpp, 215
 scheduler.h, 217
 p_lcd
 DisplayInterface, 54
 p_tempSensor
 DisplayManagement, 60
 PERIOD_MS_TASK_DISPLAY_CPU_LOAD
 DebugManagement.h, 163
 PERIOD_MS_TASK_DISPLAY_DEBUG_DATA
 DebugManagement.h, 163
 PERIOD_MS_TASK_LED
 keepAliveLed.h, 191
 PERIOD_MS_TASK_TEMP_SENSOR
 TempSensor.h, 221
 PINA_PTR
 dio_reg_atm2560.h, 174
 PINB_PTR
 dio_reg_atm2560.h, 174
 PINC_PTR
 dio_reg_atm2560.h, 175
 PIND_PTR
 dio_reg_atm2560.h, 175
 PINx_addr_mem
 dio, 42
 PINx_idx_mem
 dio, 42
 PIT_BEFORE_INVALID
 TempSensor.cpp, 220
 PORT_CNF_IN
 dio.h, 169
 PORT_CNF_OUT
 dio.h, 170
 PORT_A
 dio_port_cnf.h, 171
 PORT_B
 dio_port_cnf.h, 171
 PORT_C
 dio_port_cnf.h, 171
 PORT_D
 dio_port_cnf.h, 172
 PORTA_PTR
 dio_reg_atm2560.h, 175
 PORTB_CNF_DDRB
 dio_port_cnf.h, 172
 PORTB_CNF_PORTB
 dio_port_cnf.h, 172
 PORTB_PTR
 dio_reg_atm2560.h, 175
 PORTC_PTR
 dio_reg_atm2560.h, 175
 PORTD_PTR
 dio_reg_atm2560.h, 176
 PRESCALER_PERIODIC_TIMER
 scheduler.h, 217
 period
 scheduler::Task_t, 116
 pit_number
 scheduler, 98
 ports_init
 dio, 42
 prescaler
 timer, 130
 RS_PIN
 LCD.h, 203
 RW_PIN
 LCD.h, 203
 read
 DebugInterface, 13
 dht22, 32
 read_humidity
 TempSensor, 125
 read_temperature
 TempSensor, 125
 readTempSensor_task
 TempSensor, 122
 RefreshLine
 DisplayInterface, 50
 RemoveElement
 LinkedList, 88
 removePeriodicTask
 scheduler, 95
 RemoveWelcomeMessage_Task
 DisplayManagement, 59
 reset
 Watchdog, 141
 reset_reason
 Watchdog, 144
 ResetElementPtr
 LinkedList, 89
 SLA_ACK

I2C.h, 186
START
 I2C.h, 186
SW_PERIOD_MS
 scheduler.h, 217
sample_cnt
 CpuLoad, 9
sample_idx
 CpuLoad, 9
sample_mem
 CpuLoad, 9
scheduler, 90
 addPeriodicTask, 92
 getPitNumber, 93
 getTaskCount, 93
 LLElementCompare, 94
 launchPeriodicTasks, 94
 pit_number, 98
 removePeriodicTask, 95
 scheduler, 92
 startScheduling, 96
 task_count, 98
 Task_t, 92
 TasksLL_ptr, 98
 updateTaskPeriod, 97
scheduler.cpp, 214
 p_global_scheduler, 215
scheduler.h, 216
 p_global_scheduler, 217
PRESCALER_PERIODIC_TIMER, 217
SW_PERIOD_MS, 217
TIMER_CTC_VALUE, 217
 TaskPtr_t, 217
scheduler::Task_t, 116
 period, 116
 TaskPtr, 116
sendBool
 DebugInterface, 13
sendChar
 DebugInterface, 14
sendInteger
 DebugInterface, 15
sendString
 DebugInterface, 16, 17
setBaudRate
 uart, 132
setBitRate
 I2C, 62
SetDDRAMAddress
 LCD, 77
setInfoStringPtr
 DebugManagement, 26
setLineAlignment
 DisplayInterface, 51
setLineAlignmentAndRefresh
 DisplayInterface, 52
setTxAddress
 I2C, 63
setValidity
 TempSensor, 123
shift_data
 T_display_data, 110
shiftLine_task
 DisplayInterface, 52
size
 String, 107
startScheduling
 scheduler, 96
startTimer1
 timer, 129
stopTimer1
 timer, 130
str_cur_ptr
 T_Display_shift_data, 111
str_debug_info_message_wdg_disabled
 DebugManagement.cpp, 159
str_debug_info_message_wdg_enabled
 DebugManagement.cpp, 160
str_debug_info_message_wdg_tmo_updated
 DebugManagement.cpp, 160
str_debug_info_message_wdg_tmo_value
 DebugManagement.cpp, 160
str_debug_info_message_wrong_menu_selection
 DebugManagement.cpp, 160
str_debug_main_menu
 DebugManagement.cpp, 160
str_debug_wdg_menu
 DebugManagement.cpp, 161
str_debug_wdg_timeout_update_selection
 DebugManagement.cpp, 161
str_ptr
 T_Display_shift_data, 111
String, 99
 ~String, 100
 appendBool, 101
 appendChar, 102
 appendInteger, 103
 appendString, 103
 Clear, 104
 ComputeStringSize, 105
 getSize, 105
 getString, 106
 size, 107
 String, 100
 string, 107
string
 String, 107
String.cpp, 218
String.h, 218
SwitchWdg
 Watchdog, 142
SystemReset
 Watchdog, 143
systemReset
 DebugManagement, 26
T_ASW_init_cnf, 107

isDebugEnabled, 108
 isDisplayActivated, 108
 isLEDActivated, 108
 isTempSensorActivated, 108
T_Display_shift_data, 111
 str_cur_ptr, 111
 str_ptr, 111
 temporization, 112
T_DisplayInterface_LineAlignment
 DisplayInterface.h, 178
T_DisplayInterface_LineDisplayMode
 DisplayInterface.h, 179
T_LCD_command
 LCD.h, 203
T_LCD_conf_struct, 112
 backlight_en, 113
 cursor_en, 113
 cursorBlink_en, 113
 display_en, 113
 entryModeDir, 113
 entryModeShift, 113
 fontType_cnf, 114
 i2c_addr, 114
 i2c_bitrate, 114
 lineNumber_cnf, 114
T_LCD_config_mode
 LCD.h, 204
T_LCD_ram_area
 LCD.h, 204
T_LL_element
 LinkedList, 84
T_WDG_Reset_type
 Watchdog.h, 231
T_display_data, 109
 alignment, 109
 display_str, 110
 isEmpty, 110
 mode, 110
 shift_data, 110
TIMER_CTC_VALUE
 scheduler.h, 217
task_count
 scheduler, 98
task_period
 TempSensor, 126
Task_t
 scheduler, 92
TaskPtr
 scheduler::Task_t, 116
TaskPtr_t
 scheduler.h, 217
TasksLL_ptr
 scheduler, 98
tempDisplayString
 DisplayManagement.h, 184
TempSensor, 116
 GetHumDecimal, 118
 GetHumInteger, 119
 getHumPtr, 119
 getHumidity, 118
 getTaskPeriod, 120
 getTemp, 120
 GetTempDecimal, 121
 GetTempInteger, 121
 getTempPtr, 121
 GetValidity, 122
 read_humidity, 125
 read_temperature, 125
 readTempSensor_task, 122
 setValidity, 123
 task_period, 126
 TempSensor, 118
 updateLastValidValues, 124
 updateTaskPeriod, 124
 valid_hum, 126
 valid坑, 126
 valid_temp, 126
 validity, 126
 validity_last_read, 127
TempSensor.cpp, 219
 p_global_ASW_TempSensor, 220
 PIT_BEFORE_INVALID, 220
TempSensor.h, 220
 DHT22_PORT, 221
 p_global_ASW_TempSensor, 222
 PERIOD_MS_TASK_TEMP_SENSOR, 221
tempSensor_ptr
 DebugManagement, 29
temporization
 T_Display_shift_data, 112
timeoutUpdate
 Watchdog, 143
timer, 127
 configureTimer1, 128
 getTimer1Value, 129
 prescaler, 130
 startTimer1, 129
 stopTimer1, 130
 timer, 128
timer.cpp, 222
 p_global_BSW_timer, 223
timer.h, 223
 p_global_BSW_timer, 224
tmo_value
 Watchdog, 145
tx_address
 I2C, 64
USART_BAUDRATE
 DebugInterface.h, 157
updateLastValidValues
 TempSensor, 124
updateLineAndRefresh
 DisplayInterface, 53
updateTaskPeriod
 scheduler, 97
 TempSensor, 124

uart, 131
 BaudRate, 136
 setBaudRate, 132
 uart, 131
 uart_init, 132
 uart_read, 133
 uart_sendByte, 133
 uart_sendString, 134
 uart_transmit, 135
uart.cpp, 224
 p_global_BSW_usart, 225
uart.h, 225
 p_global_BSW_usart, 226
uart_drv_ptr
 DebugInterface, 17
uart_init
 uart, 132
uart_read
 uart, 133
uart_sendByte
 uart, 133
uart_sendString
 uart, 134
uart_transmit
 uart, 135

valid_hum
 TempSensor, 126
valid坑
 TempSensor, 126
valid_temp
 TempSensor, 126
validity
 TempSensor, 126
validity_last_read
 TempSensor, 127

WDG_TIMEOUT_DEFAULT_MS
 Watchdog.cpp, 227
WDG_TMO_120MS
 Watchdog.h, 229
WDG_TMO_15MS
 Watchdog.h, 230
WDG_TMO_1S
 Watchdog.h, 230
WDG_TMO_250MS
 Watchdog.h, 230
WDG_TMO_2S
 Watchdog.h, 230
WDG_TMO_30MS
 Watchdog.h, 230
WDG_TMO_4S
 Watchdog.h, 231
WDG_TMO_500MS
 Watchdog.h, 231
WDG_TMO_60MS
 Watchdog.h, 231
WDG_TMO_8S
 Watchdog.h, 231

Watchdog, 136
 disable, 138
 enable, 139
 extractResetReason, 139
 getResetReason, 140
 getTMOValue, 140
 isActive, 144
 isEnabled, 141
 reset, 141
 reset_reason, 144
 SwitchWdg, 142
 SystemReset, 143
 timeoutUpdate, 143
 tmo_value, 145
 Watchdog, 137

Watchdog.cpp, 226
 __attribute__, 227
 get_mcusr, 227
 p_global_BSW_wdg, 228
 WDG_TIMEOUT_DEFAULT_MS, 227

Watchdog.h, 228
 __attribute__, 232
 p_global_BSW_wdg, 232
 T_WDG_Reset_type, 231
 WDG_TMO_120MS, 229
 WDG_TMO_15MS, 230
 WDG_TMO_1S, 230
 WDG_TMO_250MS, 230
 WDG_TMO_2S, 230
 WDG_TMO_30MS, 230
 WDG_TMO_4S, 231
 WDG_TMO_500MS, 231
 WDG_TMO_60MS, 231
 WDG_TMO_8S, 231

WatchdogMenuManagement
 DebugManagement, 27

wdg_state
 debug_mgt_state_struct_t, 10

welcomeMessageString
 DisplayManagement.h, 184

write
 LCD, 78

write4bits
 LCD, 79

writeByte
 I2C, 63

WriteInRam
 LCD, 80