

Arduino

1.0

Generated by Doxygen 1.8.14

Contents

1	Hierarchical Index	1
1.1	Class Hierarchy	1
2	Class Index	3
2.1	Class List	3
3	File Index	5
3.1	File List	5
4	Class Documentation	9
4.1	Bmp180 Class Reference	9
4.1.1	Detailed Description	10
4.1.2	Constructor & Destructor Documentation	10
4.1.2.1	Bmp180()	11
4.1.3	Member Function Documentation	11
4.1.3.1	Bmp180Monitoring_Task()	11
4.1.3.2	CalculateTemperature()	12
4.1.3.3	conversionTimerInterrupt()	13
4.1.3.4	getMonitoringTaskPeriod()	14
4.1.3.5	getStatus()	14
4.1.3.6	getTemperatureValue()	15
4.1.3.7	readCalibData()	15
4.1.3.8	readChipID()	16
4.1.3.9	startNewTemperatureConversion()	17
4.1.3.10	TemperatureMonitoring()	18

4.1.4	Member Data Documentation	18
4.1.4.1	calibration_data	19
4.1.4.2	chip_id	19
4.1.4.3	i2c_drv_ptr	19
4.1.4.4	pressure_value	19
4.1.4.5	status	19
4.1.4.6	task_period	20
4.1.4.7	temperature_value	20
4.2	CpuLoad Class Reference	20
4.2.1	Detailed Description	21
4.2.2	Constructor & Destructor Documentation	21
4.2.2.1	CpuLoad()	21
4.2.3	Member Function Documentation	21
4.2.3.1	ComputeCPULoad()	21
4.2.3.2	getAverageCPULoad()	22
4.2.3.3	getCurrentCPULoad()	23
4.2.3.4	getMaxCPULoad()	23
4.2.4	Member Data Documentation	23
4.2.4.1	avg_load	24
4.2.4.2	current_load	24
4.2.4.3	last_sum_value	24
4.2.4.4	max_load	24
4.2.4.5	sample_cnt	24
4.2.4.6	sample_idx	25
4.2.4.7	sample_mem	25
4.3	debug_mgt_state_struct_t Struct Reference	25
4.3.1	Detailed Description	25
4.3.2	Member Data Documentation	25
4.3.2.1	main_state	26
4.3.2.2	wdg_state	26

4.4	DebugInterface Class Reference	26
4.4.1	Detailed Description	27
4.4.2	Constructor & Destructor Documentation	27
4.4.2.1	DebugInterface()	27
4.4.3	Member Function Documentation	28
4.4.3.1	ClearScreen()	28
4.4.3.2	nextLine()	28
4.4.3.3	read()	29
4.4.3.4	sendBool()	30
4.4.3.5	sendChar()	30
4.4.3.6	sendInteger()	31
4.4.3.7	sendString() [1/2]	32
4.4.3.8	sendString() [2/2]	33
4.4.4	Member Data Documentation	34
4.4.4.1	uart_drv_ptr	34
4.5	DebugManagement Class Reference	34
4.5.1	Detailed Description	35
4.5.2	Constructor & Destructor Documentation	36
4.5.2.1	DebugManagement()	36
4.5.3	Member Function Documentation	36
4.5.3.1	DebugModeManagement()	36
4.5.3.2	DisplayData()	37
4.5.3.3	DisplayPeriodicData_task()	38
4.5.3.4	exitDebugMenu()	39
4.5.3.5	getIftPtr()	40
4.5.3.6	getInfoStringPtr()	40
4.5.3.7	getMenuStringPtr()	41
4.5.3.8	MainMenuManagement()	41
4.5.3.9	setInfoStringPtr()	42
4.5.3.10	systemReset()	42

4.5.3.11	WatchdogMenuManagement()	43
4.5.4	Member Data Documentation	44
4.5.4.1	debug_ift_ptr	44
4.5.4.2	debug_state	44
4.5.4.3	info_string_ptr	45
4.5.4.4	isInfoStringDisplayed	45
4.5.4.5	menu_string_ptr	45
4.5.4.6	sensorMgt_ptr	45
4.6	dht22 Class Reference	46
4.6.1	Detailed Description	47
4.6.2	Constructor & Destructor Documentation	47
4.6.2.1	dht22()	47
4.6.3	Member Function Documentation	47
4.6.3.1	getHumidity()	48
4.6.3.2	getTemperature()	49
4.6.3.3	initializeCommunication()	50
4.6.3.4	read()	51
4.6.4	Member Data Documentation	52
4.6.4.1	dht22_port	52
4.6.4.2	dio_ptr	52
4.6.4.3	mem_humidity	52
4.6.4.4	mem_temperature	52
4.6.4.5	mem_validity	52
4.6.4.6	pit_last_read	53
4.7	dio Class Reference	53
4.7.1	Detailed Description	54
4.7.2	Constructor & Destructor Documentation	54
4.7.2.1	dio()	54
4.7.3	Member Function Documentation	54
4.7.3.1	dio_changePortPinCnf()	54

4.7.3.2	dio_getPort()	55
4.7.3.3	dio_getPort_fast()	56
4.7.3.4	dio_invertPort()	57
4.7.3.5	dio_memorizePINaddress()	57
4.7.3.6	dio_setPort()	58
4.7.3.7	getDDRxAddress()	59
4.7.3.8	getPINxAddress()	60
4.7.3.9	getPORTxAddress()	60
4.7.3.10	ports_init()	61
4.7.4	Member Data Documentation	61
4.7.4.1	PINx_addr_mem	62
4.7.4.2	PINx_idx_mem	62
4.8	DisplayInterface Class Reference	62
4.8.1	Detailed Description	64
4.8.2	Constructor & Destructor Documentation	64
4.8.2.1	DisplayInterface()	64
4.8.3	Member Function Documentation	64
4.8.3.1	ClearFullScreen()	65
4.8.3.2	ClearLine()	65
4.8.3.3	ClearStringInDataStruct()	66
4.8.3.4	DisplayFullLine() [1/2]	67
4.8.3.5	DisplayFullLine() [2/2]	68
4.8.3.6	FindFirstCharAddr()	69
4.8.3.7	getDisplayDataPtr()	69
4.8.3.8	IsLineEmpty()	70
4.8.3.9	RefreshLine()	70
4.8.3.10	setLineAlignment()	71
4.8.3.11	setLineAlignmentAndRefresh()	72
4.8.3.12	shiftLine_task()	72
4.8.3.13	updateLineAndRefresh()	73

4.8.4	Member Data Documentation	74
4.8.4.1	display_data	74
4.8.4.2	dummy	74
4.8.4.3	isShiftInProgress	74
4.8.4.4	p_lcd	75
4.9	DisplayManagement Class Reference	75
4.9.1	Detailed Description	76
4.9.2	Constructor & Destructor Documentation	76
4.9.2.1	DisplayManagement()	76
4.9.3	Member Function Documentation	77
4.9.3.1	DisplaySensorData_Task()	77
4.9.3.2	GetIftPointer()	78
4.9.3.3	GetSensorMgtPtr()	78
4.9.3.4	RemoveWelcomeMessage_Task()	79
4.9.4	Member Data Documentation	79
4.9.4.1	p_display_ift	79
4.9.4.2	p_SensorMgt	80
4.10	HumSensor Class Reference	80
4.10.1	Detailed Description	81
4.10.2	Constructor & Destructor Documentation	81
4.10.2.1	HumSensor() [1/2]	81
4.10.2.2	HumSensor() [2/2]	82
4.10.3	Member Function Documentation	82
4.10.3.1	readHumSensor_task()	82
4.10.3.2	updateTaskPeriod()	83
4.11	I2C Class Reference	84
4.11.1	Detailed Description	85
4.11.2	Constructor & Destructor Documentation	85
4.11.2.1	I2C()	85
4.11.3	Member Function Documentation	85

4.11.3.1 initializeBus()	86
4.11.3.2 read()	86
4.11.3.3 setBitRate()	87
4.11.3.4 write()	87
4.11.3.5 writeByte()	88
4.11.4 Member Data Documentation	89
4.11.4.1 bitrate	89
4.12 keepAliveLed Class Reference	89
4.12.1 Detailed Description	90
4.12.2 Constructor & Destructor Documentation	90
4.12.2.1 keepAliveLed()	90
4.12.3 Member Function Documentation	90
4.12.3.1 blinkLed_task()	91
4.13 LCD Class Reference	91
4.13.1 Detailed Description	93
4.13.2 Constructor & Destructor Documentation	93
4.13.2.1 LCD()	93
4.13.3 Member Function Documentation	94
4.13.3.1 command()	94
4.13.3.2 ConfigureBacklight()	95
4.13.3.3 ConfigureCursorBlink()	96
4.13.3.4 ConfigureCursorOnOff()	96
4.13.3.5 ConfigureDisplayOnOff()	97
4.13.3.6 ConfigureEntryModeDir()	97
4.13.3.7 ConfigureEntryModeShift()	98
4.13.3.8 ConfigureFontType()	99
4.13.3.9 ConfigureI2CAddr()	100
4.13.3.10 ConfigureLineNumber()	100
4.13.3.11 GetDDRAMAddress()	101
4.13.3.12 GetLineNumberCnf()	101

4.13.3.13 InitializeScreen()	102
4.13.3.14 SetDDRAMAddress()	102
4.13.3.15 write()	103
4.13.3.16 write4bits()	104
4.13.3.17 WriteInRam()	105
4.13.4 Member Data Documentation	106
4.13.4.1 backlight_enable	106
4.13.4.2 cnfCursorBlink	106
4.13.4.3 cnfCursorOnOff	106
4.13.4.4 cnfDisplayOnOff	106
4.13.4.5 cnfEntryModeDir	106
4.13.4.6 cnfEntryModeShift	107
4.13.4.7 cnfFontType	107
4.13.4.8 cnfI2C_addr	107
4.13.4.9 cnfLineNumber	107
4.13.4.10 ddrum_addr	107
4.13.4.11 i2c_drv_ptr	108
4.14 LinkedList Class Reference	108
4.14.1 Detailed Description	109
4.14.2 Member Typedef Documentation	109
4.14.2.1 T_LL_element	110
4.14.3 Constructor & Destructor Documentation	110
4.14.3.1 LinkedList()	110
4.14.3.2 ~LinkedList()	110
4.14.4 Member Function Documentation	111
4.14.4.1 AttachNewElement()	111
4.14.4.2 FindElement()	111
4.14.4.3 getCurrentElement()	112
4.14.4.4 IsLLEmpty()	113
4.14.4.5 MoveToNextElement()	113

4.14.4.6 RemoveElement()	114
4.14.4.7 ResetElementPtr()	114
4.14.5 Member Data Documentation	115
4.14.5.1 curElement_ptr	115
4.14.5.2 firstElement	115
4.15 scheduler Class Reference	115
4.15.1 Detailed Description	116
4.15.2 Member Typedef Documentation	117
4.15.2.1 Task_t	117
4.15.3 Constructor & Destructor Documentation	117
4.15.3.1 scheduler()	117
4.15.4 Member Function Documentation	117
4.15.4.1 addPeriodicTask()	117
4.15.4.2 getPitNumber()	118
4.15.4.3 getTaskCount()	119
4.15.4.4 launchPeriodicTasks()	119
4.15.4.5 LLElementCompare()	120
4.15.4.6 removePeriodicTask()	121
4.15.4.7 startScheduling()	122
4.15.4.8 updateTaskPeriod()	123
4.15.5 Member Data Documentation	124
4.15.5.1 pit_number	124
4.15.5.2 task_count	124
4.15.5.3 TasksLL_ptr	124
4.16 Sensor Class Reference	125
4.16.1 Detailed Description	126
4.16.2 Constructor & Destructor Documentation	126
4.16.2.1 Sensor() [1/2]	126
4.16.2.2 Sensor() [2/2]	126
4.16.3 Member Function Documentation	127

4.16.3.1	getRawDataPtr()	127
4.16.3.2	getTaskPeriod()	128
4.16.3.3	getValidity()	128
4.16.3.4	getValue()	128
4.16.3.5	getValueDecimal()	129
4.16.3.6	getValueInteger()	129
4.16.3.7	readSensor_task()	129
4.16.3.8	setLastValidity()	129
4.16.3.9	setValidityTMO()	130
4.16.3.10	updateTaskPeriod()	131
4.16.3.11	updateValidData()	131
4.16.4	Member Data Documentation	132
4.16.4.1	raw_data	132
4.16.4.2	task_period	132
4.16.4.3	valid_pit	132
4.16.4.4	valid_value	132
4.16.4.5	validity	132
4.16.4.6	validity_last_read	133
4.16.4.7	validity_tmo	133
4.17	SensorManagement Class Reference	133
4.17.1	Detailed Description	134
4.17.2	Constructor & Destructor Documentation	134
4.17.2.1	SensorManagement()	134
4.17.3	Member Function Documentation	134
4.17.3.1	getFullStringFormattedValue()	134
4.17.3.2	getSensorCount()	135
4.17.3.3	getSensorObjectPtr()	136
4.17.3.4	updateTaskPeriod()	137
4.17.4	Member Data Documentation	137
4.17.4.1	nb_sensors	138

4.17.4.2	sensor_ptr_table	138
4.18	String Class Reference	138
4.18.1	Detailed Description	139
4.18.2	Constructor & Destructor Documentation	139
4.18.2.1	String() [1/2]	139
4.18.2.2	String() [2/2]	140
4.18.2.3	~String()	140
4.18.3	Member Function Documentation	141
4.18.3.1	appendBool()	141
4.18.3.2	appendChar()	142
4.18.3.3	appendInteger()	142
4.18.3.4	appendSpace()	143
4.18.3.5	appendString()	144
4.18.3.6	Clear()	145
4.18.3.7	ComputeStringSize()	146
4.18.3.8	getSize()	146
4.18.3.9	getString()	147
4.18.4	Member Data Documentation	147
4.18.4.1	size	148
4.18.4.2	string	148
4.19	T_ASW_init_cnf Struct Reference	148
4.19.1	Detailed Description	148
4.19.2	Member Data Documentation	148
4.19.2.1	isDebugActivated	149
4.19.2.2	isDisplayActivated	149
4.19.2.3	isLEDActivated	149
4.19.2.4	isSensorMgtActivated	149
4.20	Bmp180::T_BMP180_calib_data Struct Reference	149
4.20.1	Detailed Description	150
4.20.2	Member Data Documentation	150

4.20.2.1	AC1	150
4.20.2.2	AC2	150
4.20.2.3	AC3	150
4.20.2.4	AC4	151
4.20.2.5	AC5	151
4.20.2.6	AC6	151
4.20.2.7	B1	151
4.20.2.8	B2	151
4.20.2.9	MB	151
4.20.2.10	MC	152
4.20.2.11	MD	152
4.21	Bmp180::T_BMP180_measurement_data Struct Reference	152
4.21.1	Detailed Description	152
4.21.2	Member Data Documentation	152
4.21.2.1	ready	152
4.21.2.2	ts	153
4.21.2.3	value	153
4.22	T_display_data Struct Reference	153
4.22.1	Detailed Description	154
4.22.2	Member Data Documentation	154
4.22.2.1	alignment	154
4.22.2.2	display_str	154
4.22.2.3	isEmpty	154
4.22.2.4	mode	154
4.22.2.5	shift_data	155
4.23	T_Display_shift_data Struct Reference	155
4.23.1	Detailed Description	155
4.23.2	Member Data Documentation	156
4.23.2.1	str_cur_ptr	156
4.23.2.2	str_ptr	156

4.23.2.3 temporization	156
4.24 T_LCD_conf_struct Struct Reference	156
4.24.1 Detailed Description	157
4.24.2 Member Data Documentation	157
4.24.2.1 backlight_en	157
4.24.2.2 cursor_en	157
4.24.2.3 cursorBlink_en	157
4.24.2.4 display_en	158
4.24.2.5 entryModeDir	158
4.24.2.6 entryModeShift	158
4.24.2.7 fontType_cnf	158
4.24.2.8 i2c_addr	158
4.24.2.9 i2c_bitrate	159
4.24.2.10 lineNumber_cnf	159
4.25 LinkedList::T_LL_element Struct Reference	159
4.25.1 Detailed Description	159
4.25.2 Member Data Documentation	160
4.25.2.1 data_ptr	160
4.25.2.2 nextElement	160
4.26 T_SensorManagement_Sensor_Config Struct Reference	160
4.26.1 Detailed Description	160
4.26.2 Member Data Documentation	161
4.26.2.1 data_name_str	161
4.26.2.2 period	161
4.26.2.3 sensor_type	161
4.26.2.4 unit_str	161
4.26.2.5 validity_tmo	161
4.27 scheduler::Task_t Struct Reference	162
4.27.1 Detailed Description	162
4.27.2 Member Data Documentation	162

4.27.2.1 period	162
4.27.2.2 TaskPtr	162
4.28 TempSensor Class Reference	163
4.28.1 Detailed Description	164
4.28.2 Constructor & Destructor Documentation	164
4.28.2.1 TempSensor() [1/2]	164
4.28.2.2 TempSensor() [2/2]	165
4.28.3 Member Function Documentation	165
4.28.3.1 readTempSensor_task()	165
4.28.3.2 updateTaskPeriod()	166
4.29 timer Class Reference	167
4.29.1 Detailed Description	168
4.29.2 Constructor & Destructor Documentation	168
4.29.2.1 timer()	168
4.29.3 Member Function Documentation	168
4.29.3.1 configureTimer1()	168
4.29.3.2 configureTimer3()	169
4.29.3.3 getTimer1Value()	170
4.29.3.4 startTimer1()	170
4.29.3.5 startTimer3()	171
4.29.3.6 stopTimer1()	171
4.29.3.7 stopTimer3()	171
4.29.4 Member Data Documentation	172
4.29.4.1 prescaler1	172
4.29.4.2 prescaler3	172
4.30 usart Class Reference	172
4.30.1 Detailed Description	173
4.30.2 Constructor & Destructor Documentation	173
4.30.2.1 usart()	173
4.30.3 Member Function Documentation	173

4.30.3.1	setBaudRate()	174
4.30.3.2	usart_init()	174
4.30.3.3	usart_read()	175
4.30.3.4	usart_sendByte()	175
4.30.3.5	usart_sendString()	176
4.30.3.6	usart_transmit()	177
4.30.4	Member Data Documentation	177
4.30.4.1	BaudRate	177
4.31	Watchdog Class Reference	177
4.31.1	Detailed Description	178
4.31.2	Constructor & Destructor Documentation	178
4.31.2.1	Watchdog() [1/2]	179
4.31.2.2	Watchdog() [2/2]	179
4.31.3	Member Function Documentation	180
4.31.3.1	disable()	180
4.31.3.2	enable()	180
4.31.3.3	getTMOValue()	181
4.31.3.4	isEnabled()	182
4.31.3.5	reset()	182
4.31.3.6	SwitchWdg()	183
4.31.3.7	SystemReset()	183
4.31.3.8	timeoutUpdate()	184
4.31.4	Member Data Documentation	185
4.31.4.1	isActive	185
4.31.4.2	tmo_value	185

5 File Documentation	187
5.1 asw.cpp File Reference	187
5.1.1 Detailed Description	188
5.1.2 Function Documentation	188
5.1.2.1 asw_init()	188
5.2 asw.h File Reference	189
5.2.1 Detailed Description	189
5.2.2 Function Documentation	189
5.2.2.1 asw_init()	190
5.3 Bmp180.cpp File Reference	190
5.3.1 Detailed Description	191
5.3.2 Variable Documentation	191
5.3.2.1 p_global_BSW_bmp180	191
5.4 Bmp180.h File Reference	192
5.4.1 Detailed Description	193
5.4.2 Macro Definition Documentation	193
5.4.2.1 BMP180_CHIP_ID_CALIB_EEP_START_ADDR	193
5.4.2.2 BMP180_CHIP_ID_EEP_ADDR	193
5.4.2.3 BMP180_CHIP_ID_EXPECTED	193
5.4.2.4 BMP180_CTRL_MEAS_EEP_ADDR	194
5.4.2.5 BMP180_CTRL_MEAS_START_TEMP_CONV	194
5.4.2.6 BMP180_I2C_ADDR	194
5.4.2.7 BMP180_I2C_BITRATE	194
5.4.2.8 BMP180_MONITORING_DEFAULT_PERIOD	194
5.4.2.9 BMP180_OUT_REG_LSB EEPROM_ADDR	195
5.4.2.10 BMP180_OUT_REG_MSB EEPROM_ADDR	195
5.4.2.11 BMP180_TEMP_MEAS_TIMER_CTC_VALUE	195
5.4.2.12 BMP180_TIMER_PRESCALER_VALUE	195
5.4.3 Enumeration Type Documentation	195
5.4.3.1 T_BMP180_status	195

5.4.4	Variable Documentation	196
5.4.4.1	p_global_BSW_bmp180	196
5.5	bsw.cpp File Reference	196
5.5.1	Detailed Description	197
5.5.2	Function Documentation	197
5.5.2.1	bsw_init()	197
5.6	bsw.h File Reference	198
5.6.1	Detailed Description	198
5.6.2	Function Documentation	198
5.6.2.1	bsw_init()	199
5.7	CpuLoad.cpp File Reference	199
5.7.1	Detailed Description	200
5.7.2	Variable Documentation	200
5.7.2.1	p_global_BSW_cpupload	200
5.8	CpuLoad.h File Reference	200
5.8.1	Detailed Description	201
5.8.2	Macro Definition Documentation	201
5.8.2.1	NB_OF_SAMPLES	201
5.8.3	Variable Documentation	201
5.8.3.1	p_global_BSW_cpupload	201
5.9	DebugInterface.cpp File Reference	202
5.9.1	Detailed Description	202
5.9.2	Variable Documentation	202
5.9.2.1	p_global_ASW_DebugInterface	202
5.10	DebugInterface.h File Reference	203
5.10.1	Detailed Description	203
5.10.2	Macro Definition Documentation	203
5.10.2.1	USART_BAUDRATE	204
5.10.3	Variable Documentation	204
5.10.3.1	p_global_ASW_DebugInterface	204

5.11 DebugManagement.cpp File Reference	204
5.11.1 Detailed Description	205
5.11.2 Variable Documentation	205
5.11.2.1 p_global_ASW_DebugManagement	205
5.11.2.2 str_debug_info_message_wdg_disabled	206
5.11.2.3 str_debug_info_message_wdg_enabled	206
5.11.2.4 str_debug_info_message_wdg_tmo_updated	206
5.11.2.5 str_debug_info_message_wdg_tmo_value	206
5.11.2.6 str_debug_info_message_wrong_menu_selection	206
5.11.2.7 str_debug_main_menu	207
5.11.2.8 str_debug_wdg_menu	207
5.11.2.9 str_debug_wdg_timeout_update_selection	207
5.12 DebugManagement.h File Reference	208
5.12.1 Detailed Description	208
5.12.2 Macro Definition Documentation	209
5.12.2.1 PERIOD_MS_TASK_DISPLAY_CPU_LOAD	209
5.12.2.2 PERIOD_MS_TASK_DISPLAY_DEBUG_DATA	209
5.12.3 Enumeration Type Documentation	209
5.12.3.1 debug_mgt_main_menu_state_t	209
5.12.3.2 debug_mgt_wdg_state_t	209
5.12.4 Variable Documentation	210
5.12.4.1 p_global_ASW_DebugManagement	210
5.13 dht22.cpp File Reference	210
5.13.1 Detailed Description	211
5.13.2 Macro Definition Documentation	211
5.13.2.1 MAX_WAIT_TIME_US	211
5.13.3 Variable Documentation	211
5.13.3.1 p_global_BSW_dht22	211
5.14 dht22.h File Reference	212
5.14.1 Detailed Description	212

5.14.2 Variable Documentation	212
5.14.2.1 p_global_BSW_dht22	212
5.15 dio.cpp File Reference	213
5.15.1 Detailed Description	213
5.15.2 Variable Documentation	213
5.15.2.1 p_global_BSW_dio	214
5.16 dio.h File Reference	214
5.16.1 Detailed Description	215
5.16.2 Macro Definition Documentation	215
5.16.2.1 DECODE_PIN	215
5.16.2.2 DECODE_PORT	215
5.16.2.3 ENCODE_PORT	215
5.16.2.4 PORT_CNF_IN	216
5.16.2.5 PORT_CNF_OUT	216
5.16.3 Variable Documentation	216
5.16.3.1 p_global_BSW_dio	216
5.17 dio_port_cnf.h File Reference	216
5.17.1 Detailed Description	217
5.17.2 Macro Definition Documentation	217
5.17.2.1 PORT_A	217
5.17.2.2 PORT_B	217
5.17.2.3 PORT_C	218
5.17.2.4 PORT_D	218
5.17.2.5 PORTB_CNF_DDRB	218
5.17.2.6 PORTB_CNF_PORTB	218
5.18 dio_reg_atm2560.h File Reference	219
5.18.1 Detailed Description	219
5.18.2 Macro Definition Documentation	219
5.18.2.1 DDRA_PTR	220
5.18.2.2 DDRB_PTR	220

5.18.2.3	DDRC_PTR	220
5.18.2.4	DDRD_PTR	220
5.18.2.5	PINA_PTR	220
5.18.2.6	PINB_PTR	221
5.18.2.7	PINC_PTR	221
5.18.2.8	PIND_PTR	221
5.18.2.9	PORTA_PTR	221
5.18.2.10	PORTB_PTR	221
5.18.2.11	PORTC_PTR	222
5.18.2.12	PORTD_PTR	222
5.19	DisplayInterface.cpp File Reference	222
5.19.1	Detailed Description	222
5.19.2	Variable Documentation	223
5.19.2.1	p_global_ASW_DisplayInterface	223
5.20	DisplayInterface.h File Reference	223
5.20.1	Detailed Description	224
5.20.2	Macro Definition Documentation	224
5.20.2.1	DISPLAY_LINE_SHIFT_PERIOD_MS	224
5.20.2.2	DISPLAY_LINE_SHIFT_TEMPO_TIME	224
5.20.3	Enumeration Type Documentation	224
5.20.3.1	T_DisplayInterface_LineAlignment	224
5.20.3.2	T_DisplayInterface_LineDisplayMode	225
5.20.4	Variable Documentation	225
5.20.4.1	p_global_ASW_DisplayInterface	225
5.21	DisplayManagement.cpp File Reference	226
5.21.1	Detailed Description	226
5.21.2	Variable Documentation	226
5.21.2.1	noSensorsDisplayString	226
5.21.2.2	p_global_ASW_DisplayManagement	227
5.21.2.3	welcomeMessageString	227

5.22 DisplayManagement.h File Reference	227
5.22.1 Detailed Description	228
5.22.2 Macro Definition Documentation	228
5.22.2.1 DISPLAY_MGT_FIRST_LINE_SENSORS	228
5.22.2.2 DISPLAY_MGT_I2C_BITRATE	228
5.22.2.3 DISPLAY_MGT_LCD_I2C_ADDR	228
5.22.2.4 DISPLAY_MGT_PERIOD_TASK_SENSOR	229
5.22.2.5 DISPLAY_MGT_PERIOD_WELCOME_MSG_REMOVAL	229
5.22.3 Variable Documentation	229
5.22.3.1 LCD_init_cnf	229
5.22.3.2 p_global_ASW_DisplayManagement	229
5.23 HumSensor.cpp File Reference	230
5.23.1 Detailed Description	230
5.23.2 Macro Definition Documentation	230
5.23.2.1 DHT22_PORT	230
5.24 HumSensor.h File Reference	231
5.24.1 Detailed Description	231
5.25 I2C.cpp File Reference	231
5.25.1 Detailed Description	232
5.25.2 Variable Documentation	232
5.25.2.1 p_global_BSW_i2c	232
5.26 I2C.h File Reference	232
5.26.1 Detailed Description	233
5.26.2 Macro Definition Documentation	233
5.26.2.1 DATA_ACK	233
5.26.2.2 REPEATED_START	233
5.26.2.3 SLAR_ACK	234
5.26.2.4 SLAW_ACK	234
5.26.2.5 START	234
5.26.3 Variable Documentation	234

5.26.3.1 p_global_BSW_i2c	234
5.27 int.cpp File Reference	235
5.27.1 Detailed Description	235
5.27.2 Function Documentation	235
5.27.2.1 ISR() [1/3]	236
5.27.2.2 ISR() [2/3]	236
5.27.2.3 ISR() [3/3]	237
5.28 keepAliveLed.cpp File Reference	238
5.28.1 Detailed Description	238
5.28.2 Variable Documentation	238
5.28.2.1 p_global_ASW_keepAliveLed	238
5.29 keepAliveLed.h File Reference	239
5.29.1 Detailed Description	239
5.29.2 Macro Definition Documentation	239
5.29.2.1 LED_PORT	240
5.29.2.2 PERIOD_MS_TASK_LED	240
5.29.3 Variable Documentation	240
5.29.3.1 p_global_ASW_keepAliveLed	240
5.30 LCD.cpp File Reference	240
5.30.1 Detailed Description	241
5.30.2 Variable Documentation	241
5.30.2.1 p_global_BSW_lcd	241
5.31 LCD.h File Reference	241
5.31.1 Detailed Description	243
5.31.2 Macro Definition Documentation	243
5.31.2.1 BACKLIGHT_PIN	243
5.31.2.2 EN_PIN	243
5.31.2.3 LCD_CNF_BACKLIGHT_OFF	243
5.31.2.4 LCD_CNF_BACKLIGHT_ON	244
5.31.2.5 LCD_CNF_CURSOR_BLINK_OFF	244

5.31.2.6 LCD_CNF_CURSOR_BLINK_ON	244
5.31.2.7 LCD_CNF_CURSOR_OFF	244
5.31.2.8 LCD_CNF_CURSOR_ON	244
5.31.2.9 LCD_CNF_DISPLAY_OFF	245
5.31.2.10 LCD_CNF_DISPLAY_ON	245
5.31.2.11 LCD_CNF_ENTRY_MODE_DIRECTION_LEFT	245
5.31.2.12 LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT	245
5.31.2.13 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF	245
5.31.2.14 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON	246
5.31.2.15 LCD_CNF_FONT_5_11	246
5.31.2.16 LCD_CNF_FONT_5_8	246
5.31.2.17 LCD_CNF_ONE_LINE	246
5.31.2.18 LCD_CNF_SHIFT_ID	246
5.31.2.19 LCD_CNF_SHIFT_SH	247
5.31.2.20 LCD_CNF_TWO_LINE	247
5.31.2.21 LCD_DISPLAY_CTRL_FIELD_B	247
5.31.2.22 LCD_DISPLAY_CTRL_FIELD_C	247
5.31.2.23 LCD_DISPLAY_CTRL_FIELD_D	247
5.31.2.24 LCD_FCT_SET_FIELD_DL	248
5.31.2.25 LCD_FCT_SET_FIELD_F	248
5.31.2.26 LCD_FCT_SET_FIELD_N	248
5.31.2.27 LCD_INST_CLR_DISPLAY_BIT	248
5.31.2.28 LCD_INST_DISPLAY_CTRL	248
5.31.2.29 LCD_INST_ENTRY_MODE_SET	249
5.31.2.30 LCD_INST_FUNCTION_SET	249
5.31.2.31 LCD_INST_SET_DDRAM_ADDR	249
5.31.2.32 LCD_RAM_1_LINE_MAX	249
5.31.2.33 LCD_RAM_1_LINE_MIN	249
5.31.2.34 LCD_RAM_2_LINES_MAX_1	250
5.31.2.35 LCD_RAM_2_LINES_MAX_2	250

5.31.2.36 LCD_RAM_2_LINES_MIN_1	250
5.31.2.37 LCD_RAM_2_LINES_MIN_2	250
5.31.2.38 LCD_SIZE_NB_CHAR_PER_LINE	250
5.31.2.39 LCD_SIZE_NB_LINES	251
5.31.2.40 LCD_WAIT_CLR_RETURN	251
5.31.2.41 LCD_WAIT_OTHER_MODES	251
5.31.2.42 RS_PIN	251
5.31.2.43 RW_PIN	251
5.31.3 Enumeration Type Documentation	251
5.31.3.1 T_LCD_command	251
5.31.3.2 T_LCD_config_mode	252
5.31.3.3 T_LCD_ram_area	252
5.31.4 Variable Documentation	252
5.31.4.1 p_global_BSW_lcd	253
5.32 LinkedList.cpp File Reference	253
5.32.1 Detailed Description	253
5.33 LinkedList.h File Reference	253
5.33.1 Detailed Description	254
5.33.2 Typedef Documentation	254
5.33.2.1 CompareFctPtr_t	254
5.34 main.cpp File Reference	254
5.34.1 Detailed Description	255
5.34.2 Macro Definition Documentation	255
5.34.2.1 DEBUG_ACTIVE_PORT	255
5.34.3 Function Documentation	255
5.34.3.1 main()	256
5.34.4 Variable Documentation	256
5.34.4.1 ASW_init_cnf	256
5.34.4.2 isDebugModeActivated	256
5.35 main.h File Reference	257

5.35.1	Detailed Description	257
5.35.2	Variable Documentation	257
5.35.2.1	ASW_init_cnf	257
5.35.2.2	isDebugModeActivated	258
5.36	operators.cpp File Reference	258
5.36.1	Detailed Description	258
5.36.2	Function Documentation	259
5.36.2.1	operator delete()	259
5.36.2.2	operator new()	259
5.37	operators.h File Reference	259
5.37.1	Detailed Description	260
5.37.2	Function Documentation	260
5.37.2.1	operator delete()	260
5.37.2.2	operator new()	261
5.38	scheduler.cpp File Reference	261
5.38.1	Detailed Description	262
5.38.2	Variable Documentation	262
5.38.2.1	p_global_scheduler	262
5.39	scheduler.h File Reference	262
5.39.1	Detailed Description	263
5.39.2	Macro Definition Documentation	263
5.39.2.1	PRESCALER_PERIODIC_TIMER	263
5.39.2.2	SW_PERIOD_MS	263
5.39.2.3	TIMER_CTC_VALUE	264
5.39.3	Typedef Documentation	264
5.39.3.1	TaskPtr_t	264
5.39.4	Variable Documentation	264
5.39.4.1	p_global_scheduler	264
5.40	Sensor.cpp File Reference	264
5.40.1	Detailed Description	265

5.40.2 Macro Definition Documentation	265
5.40.2.1 TASK_PERIOD_DEFAULT	265
5.40.2.2 VALIDITY_TIMEOUT_MS_DEFAULT	265
5.41 Sensor.h File Reference	265
5.41.1 Detailed Description	266
5.42 sensor_configuration.cpp File Reference	266
5.42.1 Detailed Description	267
5.42.2 Macro Definition Documentation	267
5.42.2.1 SENSOR_MGT_CNF_DEFAULT_PERIOD	267
5.42.2.2 SENSOR_MGT_CNF_DEFAULT_TMO	267
5.42.3 Variable Documentation	267
5.42.3.1 SensorManagement_Sensor_Config_list	268
5.43 sensor_configuration.h File Reference	268
5.43.1 Detailed Description	269
5.43.2 Variable Documentation	269
5.43.2.1 SensorManagement_Sensor_Config_list	269
5.44 SensorManagement.cpp File Reference	269
5.44.1 Detailed Description	270
5.44.2 Variable Documentation	270
5.44.2.1 p_global_ASW_SensorManagement	270
5.45 SensorManagement.h File Reference	270
5.45.1 Detailed Description	271
5.45.2 Enumeration Type Documentation	271
5.45.2.1 T_SensorManagement_Sensor_Type	271
5.45.3 Variable Documentation	271
5.45.3.1 p_global_ASW_SensorManagement	271
5.46 String.cpp File Reference	272
5.46.1 Detailed Description	272
5.47 String.h File Reference	272
5.47.1 Detailed Description	273

5.48 TempSensor.cpp File Reference	273
5.48.1 Detailed Description	273
5.48.2 Macro Definition Documentation	274
5.48.2.1 DHT22_PORT	274
5.48.2.2 TEMP_SENSOR_VALUE_ACCEPTABLE_DIFFERENCE	274
5.49 TempSensor.h File Reference	274
5.49.1 Detailed Description	275
5.50 timer.cpp File Reference	275
5.50.1 Detailed Description	275
5.50.2 Variable Documentation	276
5.50.2.1 p_global_BSW_timer	276
5.51 timer.h File Reference	276
5.51.1 Detailed Description	276
5.51.2 Variable Documentation	277
5.51.2.1 p_global_BSW_timer	277
5.52 usart.cpp File Reference	277
5.52.1 Detailed Description	277
5.52.2 Variable Documentation	278
5.52.2.1 p_global_BSW_usart	278
5.53 usart.h File Reference	278
5.53.1 Detailed Description	278
5.53.2 Variable Documentation	279
5.53.2.1 p_global_BSW_usart	279
5.54 Watchdog.cpp File Reference	279
5.54.1 Detailed Description	280
5.54.2 Macro Definition Documentation	280
5.54.2.1 WDG_TIMEOUT_DEFAULT_MS	280
5.54.3 Variable Documentation	280
5.54.3.1 p_global_BSW_wdg	280
5.55 Watchdog.h File Reference	280

5.55.1 Detailed Description	281
5.55.2 Macro Definition Documentation	281
5.55.2.1 WDG_TMO_120MS	281
5.55.2.2 WDG_TMO_15MS	282
5.55.2.3 WDG_TMO_1S	282
5.55.2.4 WDG_TMO_250MS	282
5.55.2.5 WDG_TMO_2S	282
5.55.2.6 WDG_TMO_30MS	282
5.55.2.7 WDG_TMO_4S	283
5.55.2.8 WDG_TMO_500MS	283
5.55.2.9 WDG_TMO_60MS	283
5.55.2.10 WDG_TMO_8S	283
5.55.3 Variable Documentation	283
5.55.3.1 p_global_BSW_wdg	283
Index	285

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Bmp180	9
CpuLoad	20
debug_mgt_state_struct_t	25
DebugInterface	26
DebugManagement	34
dht22	46
dio	53
DisplayInterface	62
DisplayManagement	75
I2C	84
keepAliveLed	89
LCD	91
LinkedList	108
scheduler	115
Sensor	125
HumSensor	80
TempSensor	163
SensorManagement	133
String	138
T_ASW_init_cnf	148
Bmp180::T_BMP180_calib_data	149
Bmp180::T_BMP180_measurement_data	152
T_display_data	153
T_Display_shift_data	155
T_LCD_conf_struct	156
LinkedList::T_LL_element	159
T_SensorManagement_Sensor_Config	160
scheduler::Task_t	162
timer	167
uart	172
Watchdog	177

Chapter 2

Class Index

2.1 Class List

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

Bmp180	BMP180 sensor class definition	9
CpuLoad	Class defining CPU load libraries	20
debug_mgt_state_struct_t	Structure containing all debug states	25
DebugInterface	Class used for debugging on usart link	26
DebugManagement	Debug management class	34
dht22	DHT 22 driver class	46
dio	DIO class	53
DisplayInterface	Display interface services class	62
DisplayManagement	Display management class	75
HumSensor	Class for humidity sensor	80
I2C	Two-wire serial interface (I2C) class definition	84
keepAliveLed	Class for keep-alive LED blinking	89
LCD	Class for LCD S2004A display driver	91
LinkedList	Linked list class	108
scheduler	Scheduler class	115
Sensor	Generic class for sensor device	125
SensorManagement	Sensor management class	133
String	String management class	138

T_ASW_init_cnf	ASW initialization configuration structure	148
Bmp180::T_BMP180_calib_data	Structure defining the calibration data of BMP180 sensor	149
Bmp180::T_BMP180_measurement_data	Structure defining a sensor value and its status	152
T_display_data	Structure containing display data	153
T_Display_shift_data	Structure containing shift data	155
T_LCD_conf_struct	Structure defining LCD configuration	156
LinkedList::T_LL_element	Type defining a linked list element	159
T_SensorManagement_Sensor_Config	Sensor informations structure	160
scheduler::Task_t	Type defining a task structure	162
TempSensor	Class for temperature sensor	163
timer	Class defining a timer	167
uart	USART serial bus class	172
Watchdog	Watchdog management class	177

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

asw.cpp	ASW main file	187
asw.h	ASW main header file	189
Bmp180.cpp	Bmp180 class source file	190
Bmp180.h	Bmp180 class header file	192
bsw.cpp	BSW main file	196
bsw.h	BSW main header file	198
CpuLoad.cpp	Defines functions of class CpuLoad	199
CpuLoad.h	CpuLoad class header file	200
DebugInterface.cpp	This file defines classes for log and debug data transmission on USART link	202
DebugInterface.h	Header file for debug and logging functions	203
DebugManagement.cpp	Debug management class source file	204
DebugManagement.h	Debug management class header file	208
dht22.cpp	This file defines classes for DHT22 driver	210
dht22.h	DHT22 driver header file	212
dio.cpp	DIO library	213
dio.h	DIO library header file	214
dio_port_cnf.h	Digital ports configuration file	216
dio_reg_atm2560.h	Defines DIO register addresses for ATMEGA2560	219

DisplayInterface.cpp	Source code file for display services	222
DisplayInterface.h	DisplayInterface class header file	223
DisplayManagement.cpp	Display management source file	226
DisplayManagement.h	Display management class header file	227
HumSensor.cpp	Defines function of class HumSensor	230
HumSensor.h	Class HumSensor header file	231
I2C.cpp	Two-wire interface (I2C) source file	231
I2C.h	I2C class header file	232
int.cpp	Interrupt management source file	235
keepAliveLed.cpp	Definition of function for class keepAliveLed	238
keepAliveLed.h	Class keepAliveLed header file	239
LCD.cpp	LCD class source file	240
LCD.h	LCD class header file	241
LinkedList.cpp	Linked List library source file	253
LinkedList.h	Linked List library header file	253
main.cpp	Background task file	254
main.h	Background task header file	257
operators.cpp	C++ operators definitions	258
operators.h	C++ operators definitions header file	259
scheduler.cpp	Defines scheduler class	261
scheduler.h	Scheduler class header file	262
Sensor.cpp	Sensor class source code file	264
Sensor.h	Sensor class header file	265
sensor_configuration.cpp	Sensor configuration file	266
sensor_configuration.h	Sensors configuration header file	268
SensorManagement.cpp	SensorManagement class source code file	269
SensorManagement.h	SensorManagement class header file	270
String.cpp	String class source file	272
String.h	String class header file	272

TempSensor.cpp	Defines function of class TempSensor	273
TempSensor.h	Class TempSensor header file	274
timer.cpp	Defines function for class timer	275
timer.h	Timer class header file	276
usart.cpp	BSW library for USART	277
usart.h	Header file for USART library	278
Watchdog.cpp	Class Watchdog source code file	279
Watchdog.h	Class Watchdog header file	280

Chapter 4

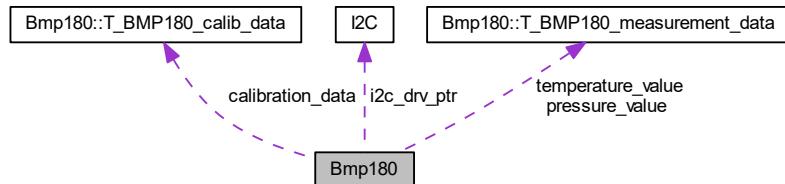
Class Documentation

4.1 Bmp180 Class Reference

BMP180 sensor class definition.

```
#include <Bmp180.h>
```

Collaboration diagram for Bmp180:



Classes

- struct [T_BMP180_calib_data](#)
Structure defining the calibration data of BMP180 sensor.
- struct [T_BMP180_measurement_data](#)
Structure defining a sensor value and its status.

Public Member Functions

- [Bmp180 \(\)](#)
Bmp180 class constructor.
- bool [getTemperatureValue \(uint16_t *data\)](#)
Temperature reading function.
- void [conversionTimerInterrupt \(\)](#)
End of conversion interrupt.

- `T_BMP180_status getStatus ()`
Driver status get function.
- `uint16_t getMonitoringTaskPeriod ()`
Task period get function.
- `void startNewTemperatureConversion ()`
Starts a new temperature conversion.
- `void TemperatureMonitoring ()`
Temperature value monitoring function.

Static Public Member Functions

- `static void Bmp180Monitoring_Task ()`
BMP180 periodic monitoring function.

Private Member Functions

- `void readCalibData ()`
Calibration data reading function.
- `void readChipID ()`
Chip ID read function.
- `void CalculateTemperature (uint16_t UT)`
Temperature calculation function.

Private Attributes

- `I2C * i2c_drv_ptr`
- `uint8_t chip_id`
- `T_BMP180_status status`
- `uint16_t task_period`
- `T_BMP180_calib_data calibration_data`
- `T_BMP180_measurement_data temperature_value`
- `T_BMP180_measurement_data pressure_value`

4.1.1 Detailed Description

BMP180 sensor class definition.

This class manages BMP180 driver.

Definition at line 50 of file Bmp180.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Bmp180()

```
Bmp180::Bmp180 ( )
```

Bmp180 class constructor.

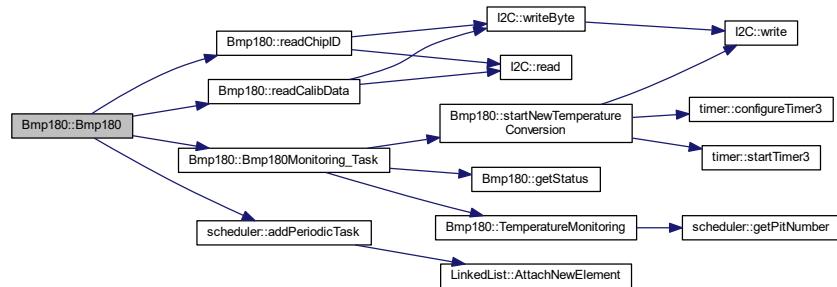
This function initializes the class **Bmp180**. It reads the chip ID and reads calibration data. If the chip ID or calibration data are incorrect, the status is set to communication failed. It also starts the periodic monitoring of the driver.

Returns

Nothing

Definition at line 23 of file Bmp180.cpp.

Here is the call graph for this function:



4.1.3 Member Function Documentation

4.1.3.1 Bmp180Monitoring_Task()

```
void Bmp180::Bmp180Monitoring_Task ( ) [static]
```

BMP180 periodic monitoring function.

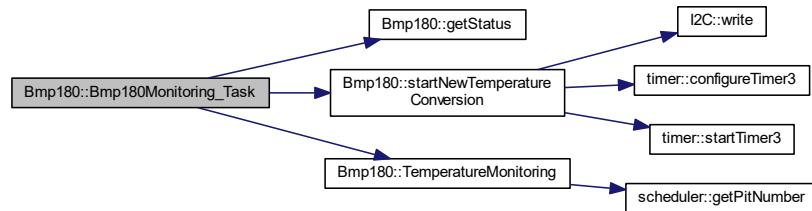
This function is in charge of monitoring the BMP180 sensor. It starts new conversions of temperature and pressure values. Temperature and pressure values time stamps are monitored and availability of measures are updated. It also monitors the status of the driver, if the driver is failed, it tries to restart the sensor device.

Returns

Nothing

Definition at line 213 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.1.3.2 CalculateTemperature()**

```
void Bmp180::CalculateTemperature (
    uint16_t UT ) [private]
```

Temperature calculation function.

This function calculates the true temperature from the raw value from sensor according to the BMP180 datasheet. The true temperature value is stored in `temperature_value` structure.

Parameters

in	<i>UT</i>	Raw temperature value
----	-----------	-----------------------

Returns

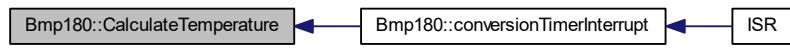
Nothing

Definition at line 203 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.3 conversionTimerInterrupt()

```
void Bmp180::conversionTimerInterrupt ( )
```

End of conversion interrupt.

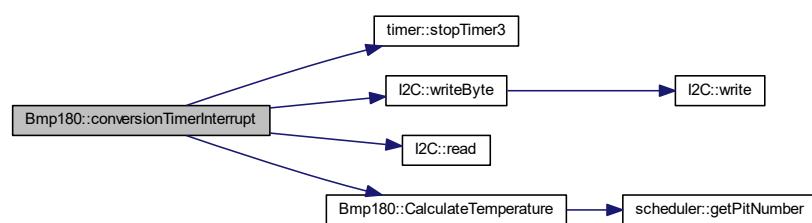
This function is called by the timer interrupt at the end of the conversion. It will retrieve the raw temperature or pressure value according to the conversion type and then calculate true value.

Returns

Nothing

Definition at line 154 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.4 getMonitoringTaskPeriod()

```
uint16_t Bmp180::getMonitoringTaskPeriod ( ) [inline]
```

Task period get function.

This function returns the task period of the monitoring function.

Returns

Task period

Definition at line 114 of file Bmp180.h.

4.1.3.5 getStatus()

```
T_BMP180_STATUS Bmp180::getStatus ( ) [inline]
```

Driver status get function.

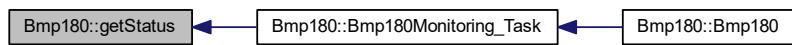
This function returns the status of the driver.

Returns

Driver status

Definition at line 103 of file Bmp180.h.

Here is the caller graph for this function:



4.1.3.6 getTemperatureValue()

```
bool Bmp180::getTemperatureValue (   
    uint16_t * data )
```

Temperature reading function.

This function is used to read the temperature value from BMP180 sensor. If a temperature measurement is available, this value is returned directly. A new conversion is started after the read operation. The result of this conversion will be retrieved by an interrupt after 4.5 ms. If the driver status is not OK, the function behaves as if no measurement is available.

Parameters

in	<code>data</code>	Pointer the location where the temperature data shall be stored.
----	-------------------	--

Returns

True if a temperature measurement is available, false otherwise.

Definition at line 126 of file Bmp180.cpp.

Here is the caller graph for this function:



4.1.3.7 readCalibData()

```
void Bmp180::readCalibData ( ) [private]
```

Calibration data reading function.

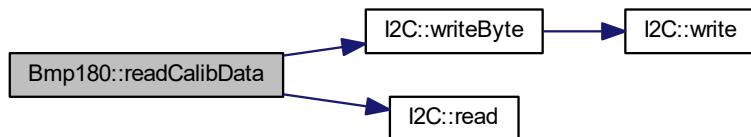
This function reads the pressure and temperature calibration data in BMP180 EEPROM using [I2C](#) bus. It also updates the driver status if the communication with the device is failed.

Returns

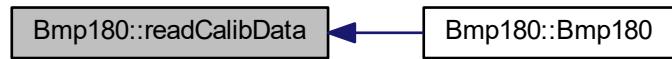
Nothing.

Definition at line 61 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.8 `readChipID()`

```
void Bmp180::readChipID( ) [private]
```

Chip ID read function.

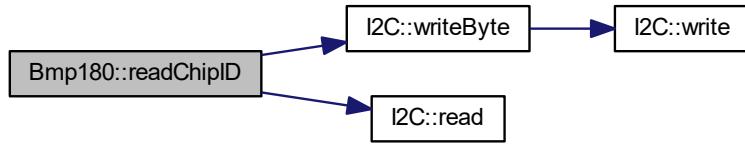
This function reads the ID of the sensor chip using `I2C` bus. It also updates the driver status if the communication is failed.

Returns

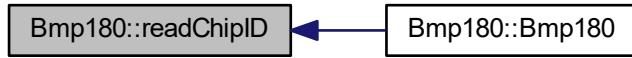
Nothing

Definition at line 107 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.9 startNewTemperatureConversion()

```
void Bmp180::startNewTemperatureConversion ( )
```

Starts a new temperature conversion.

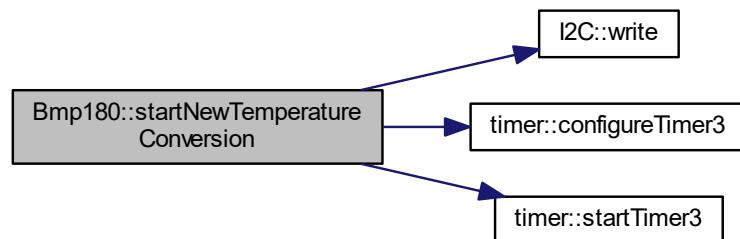
This function starts a new temperature conversion by writing 0x2E into register 0xF4 of sensor. It also starts a timer to retrieve sensor data after the conversion time.

Returns

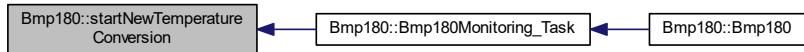
Nothing

Definition at line 133 of file `Bmp180.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.10 TemperatureMonitoring()

```
void Bmp180::TemperatureMonitoring( )
```

Temperature value monitoring function.

This function monitors the temperature value. If the last correct measurement was done more than twice the monitoring period in the past, the value is declared not ready.

Returns

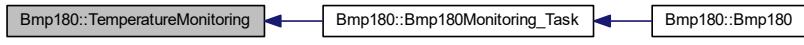
Nothing

Definition at line 223 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.4 Member Data Documentation

4.1.4.1 calibration_data

`T_BMP180_calib_data Bmp180::calibration_data [private]`

Calibration data of the sensor

Definition at line 163 of file Bmp180.h.

4.1.4.2 chip_id

`uint8_t Bmp180::chip_id [private]`

Sensor chip ID

Definition at line 140 of file Bmp180.h.

4.1.4.3 i2c_drv_ptr

`I2C* Bmp180::i2c_drv_ptr [private]`

Pointer to the **I2C** driver object

Definition at line 139 of file Bmp180.h.

4.1.4.4 pressure_value

`T_BMP180_measurement_data Bmp180::pressure_value [private]`

Pressure data structure

Definition at line 177 of file Bmp180.h.

4.1.4.5 status

`T_BMP180_status Bmp180::status [private]`

Sensor status

Definition at line 141 of file Bmp180.h.

4.1.4.6 task_period

`uint16_t Bmp180::task_period [private]`

Period of the monitoring task

Definition at line 142 of file Bmp180.h.

4.1.4.7 temperature_value

`T_BMP180_measurement_data Bmp180::temperature_value [private]`

Temperature data structure

Definition at line 176 of file Bmp180.h.

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

- [Bmp180.h](#)
- [Bmp180.cpp](#)

4.2 CpuLoad Class Reference

Class defining CPU load libraries.

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

Public Member Functions

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

Private Attributes

- `uint8_t current_load`
- `uint8_t avg_load`
- `uint8_t max_load`
- `uint8_t sample_cnt`
- `uint8_t sample_mem [NB_OF_SAMPLES]`
- `uint8_t sample_idx`
- `uint16_t last_sum_value`

4.2.1 Detailed Description

Class defining CPU load libraries.

This class defines tools to compute and monitor CPU load.

Definition at line 19 of file CpuLoad.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 CpuLoad()

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

[CpuLoad](#) class constructor.

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

Returns

Nothing

Definition at line 20 of file CpuLoad.cpp.

4.2.3 Member Function Documentation

4.2.3.1 ComputeCPULoad()

```
void CpuLoad::ComputeCPULoad ( )
```

Computes current CPU load.

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

Returns

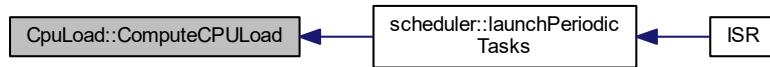
Nothing

Definition at line 40 of file CpuLoad.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.3.2 getAverageCPUload()

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

Get average CPU load value.

This function returns the average CPU load value

Returns

Average CPU load value

Definition at line 58 of file CpuLoad.h.

Here is the caller graph for this function:



4.2.3.3 getCurrrentCPULoad()

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

Get current CPU load value.

This function returns the current CPU load value

Returns

Current CPU load value

Definition at line 47 of file CpuLoad.h.

Here is the caller graph for this function:



4.2.3.4 getMaxCPULoad()

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

Get maximum CPU load value.

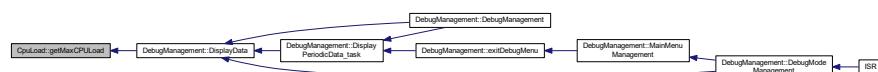
This function returns the maximum CPU load value

Returns

Maximum CPU load value

Definition at line 69 of file CpuLoad.h.

Here is the caller graph for this function:



4.2.4 Member Data Documentation

4.2.4.1 avg_load

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

Average CPU load based on the last 50 cycles

Definition at line 76 of file CpuLoad.h.

4.2.4.2 current_load

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

Current CPU load (load of last cycle)

Definition at line 75 of file CpuLoad.h.

4.2.4.3 last_sum_value

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

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

Definition at line 81 of file CpuLoad.h.

4.2.4.4 max_load

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

Maximum CPU load since power on

Definition at line 77 of file CpuLoad.h.

4.2.4.5 sample_cnt

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

Number of samples used to compute average load

Definition at line 78 of file CpuLoad.h.

4.2.4.6 sample_idx

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

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

Definition at line 80 of file CpuLoad.h.

4.2.4.7 sample_mem

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

Memorization of the last NB_OF_SAMPLES measures

Definition at line 79 of file CpuLoad.h.

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

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

4.3 debug_mgt_state_struct_t Struct Reference

Structure containing all debug states.

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

Public Attributes

- [debug_mgt_main_menu_state_t main_state](#)
- [debug_mgt_wdg_state_t wdg_state](#)

4.3.1 Detailed Description

Structure containing all debug states.

Definition at line 40 of file DebugManagement.h.

4.3.2 Member Data Documentation

4.3.2.1 main_state

```
debug_mgt_main_menu_state_t debug_mgt_state_struct_t::main_state
```

Current main menu state

Definition at line 42 of file DebugManagement.h.

4.3.2.2 wdg_state

```
debug_mgt_wdg_state_t debug_mgt_state_struct_t::wdg_state
```

Current state of watchdog management

Definition at line 43 of file DebugManagement.h.

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

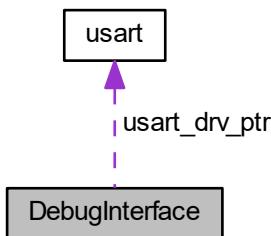
- [DebugManagement.h](#)

4.4 DebugInterface Class Reference

Class used for debugging on usart link.

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

Collaboration diagram for DebugInterface:



Public Member Functions

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

Private Attributes

- [uart * usart_drv_ptr](#)

4.4.1 Detailed Description

Class used for debugging on usart link.

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

Definition at line 21 of file DebugInterface.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 DebugInterface()

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

Class DebugInterface constructor.

Initializes the class DebugInterface. It creates a new instance of USART driver of needed.

Returns

Nothing

Definition at line 22 of file DebugInterface.cpp.

4.4.3 Member Function Documentation

4.4.3.1 ClearScreen()

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

Screen clearing function.

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

Returns

Nothing

Definition at line 77 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.4.3.2 nextLine()

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

Go to next line function.

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

Returns

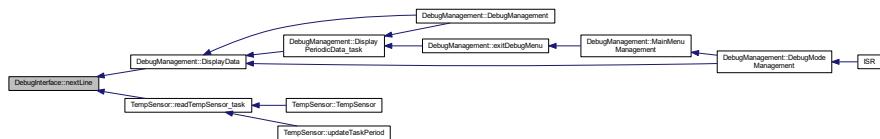
Nothing

Definition at line 71 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.4.3.3 read()**

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

USART read function.

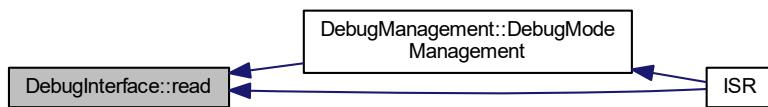
This function will read the last received byte on USART link

Returns

Received byte

Definition at line 82 of file DebugInterface.h.

Here is the caller graph for this function:



4.4.3.4 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

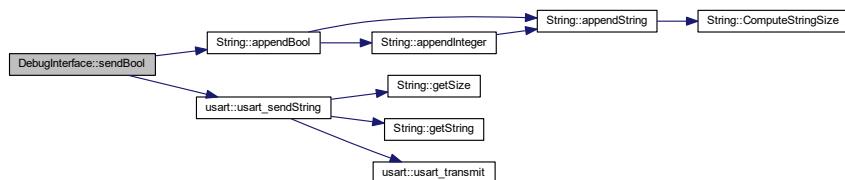
in	<code>data</code>	boolean data to be sent
in	<code>isText</code>	String conversion configuration

Returns

Nothing

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

Here is the call graph for this function:



4.4.3.5 sendChar()

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

Send a single character on USART link.

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

Parameters

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

Returns

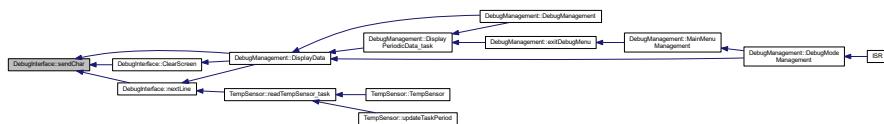
Nothing

Definition at line 44 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.4.3.6 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

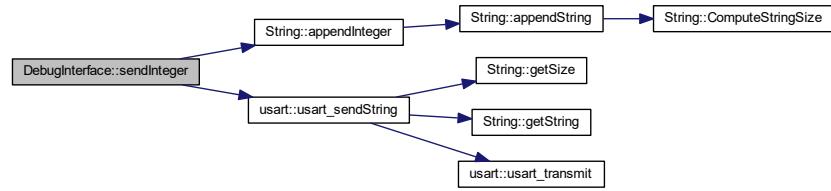
in	data	integer data to be sent
in	base	numerical base used to convert integer into string (between 2 and 36)

Returns

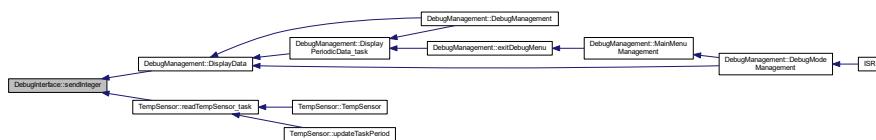
Nothing

Definition at line 49 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.4.3.7 sendString() [1/2]

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

Send a string on USART link.

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

Parameters

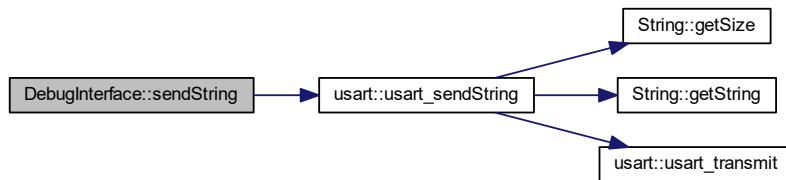
in	<i>str</i>	Pointer to the string being sent
----	------------	----------------------------------

Returns

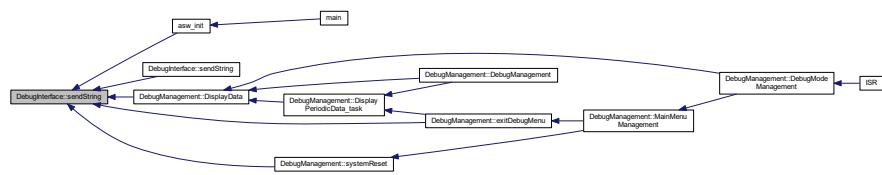
Nothing

Definition at line 31 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.4.3.8 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

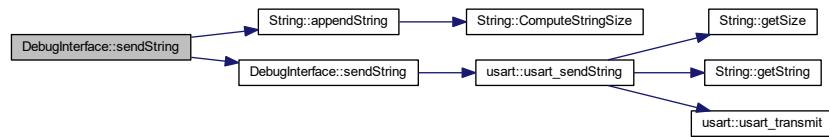
<code>in</code>	<code>str</code>	Pointer to the chain to send.
-----------------	------------------	-------------------------------

Returns

Nothing

Definition at line 37 of file DebugInterface.cpp.

Here is the call graph for this function:



4.4.4 Member Data Documentation

4.4.4.1 usart_drv_ptr

`usart* DebugInterface::usart_drv_ptr [private]`

Pointer to USART driver object

Definition at line 107 of file DebugInterface.h.

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

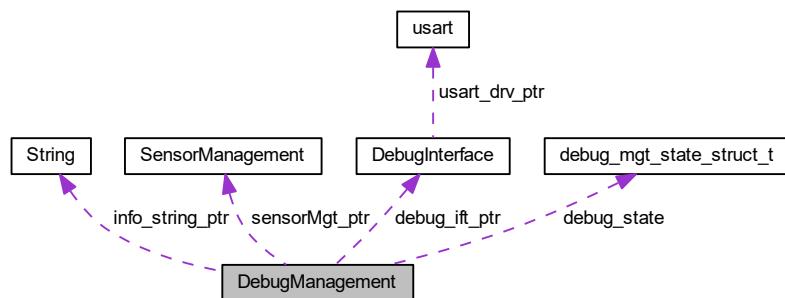
- [DebugInterface.h](#)
- [DebugInterface.cpp](#)

4.5 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 * getIfPtr ()**
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_if_ptr**
- **SensorManagement * sensorMgt_ptr**
- **uint8_t * menu_string_ptr**
- **String * info_string_ptr**
- **debug_mgt_state_struct_t debug_state**
- **bool isInfoStringDisplayed**

4.5.1 Detailed Description

Debug management class.

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

Definition at line 51 of file DebugManagement.h.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 DebugManagement()

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

Class constructor.

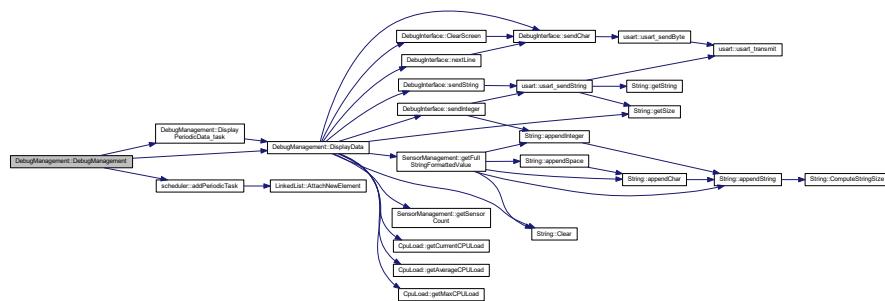
This function initializes the class. If needed, it creates a new instance of debug interface object.

Returns

Nothing

Definition at line 103 of file DebugManagement.cpp.

Here is the call graph for this function:



4.5.3 Member Function Documentation

4.5.3.1 DebugModeManagement()

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

Management of debug mode.

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

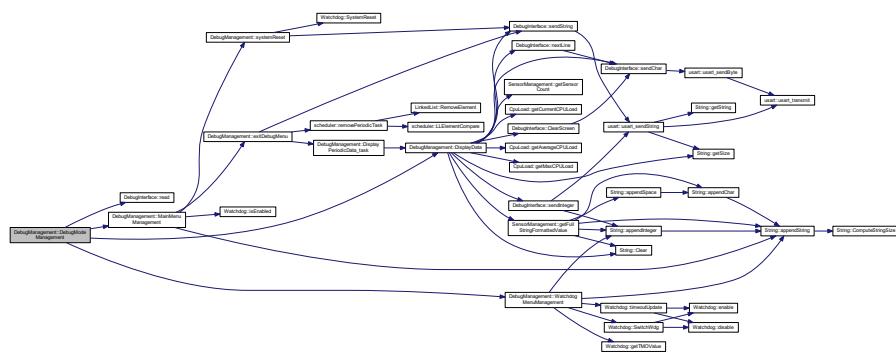
- MAIN_MENU state : handles user choice in main menu and selects next state
- WDG_MENU state : handles user choice in watchdog menu and selects next state
It is called each time a data is received on USART and debug mode is active.

Returns

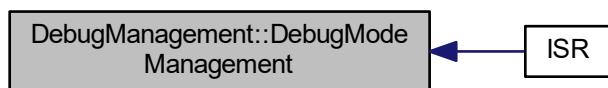
True if the debug mode shall be closed, false otherwise

Definition at line 203 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.5.3.2 DisplayData()

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

Displays data on usart link.

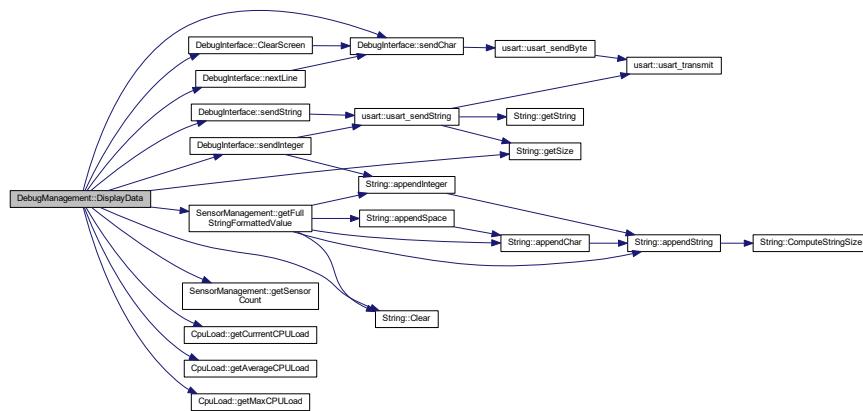
This task displays the menu and periodic data (temperature, humidity and CPU load) on usart screen.

Returns

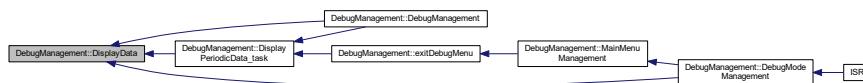
Nothing

Definition at line 132 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.5.3.3 DisplayPeriodicData_task()

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

Displays periodic data on uart link.

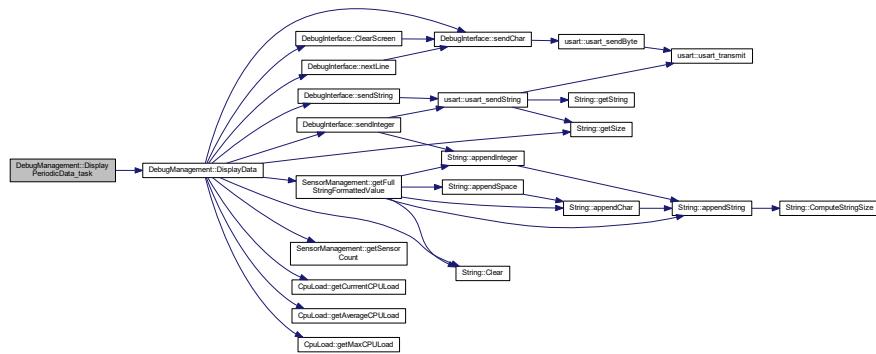
This task displays the menu and periodic data (temperature, humidity and CPU load) on uart screen. It only calls the function DisplayData.

Returns

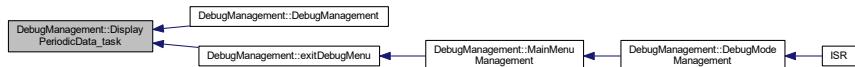
Nothing

Definition at line 195 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.5.3.4 exitDebugMenu()**

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

Debug menu exit function.

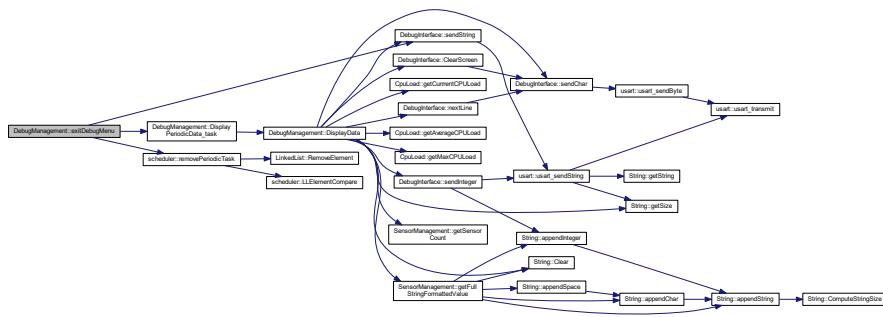
This function prepares the exit of debug menu. It writes the message "Bye !" on the screen and removes the periodic task from the scheduler.

Returns

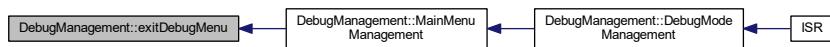
Nothing.

Definition at line 230 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.5.3.5 getIfpt()

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

Interface pointer get function.

This function returns the pointer to the debug interface object

Returns

Pointer to debug interface

Definition at line 95 of file DebugManagement.h.

4.5.3.6 getInfoStringPtr()

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

Info string get function.

This function returns the pointer to the info string to display

Returns

Info string pointer

Definition at line 115 of file DebugManagement.h.

4.5.3.7 getMenuStringPtr()

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

Menu string get function.

This function returns the pointer to the menu string to display

Returns

Menu string pointer

Definition at line 105 of file DebugManagement.h.

4.5.3.8 MainMenuManagement()

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

Main menu management.

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

Parameters

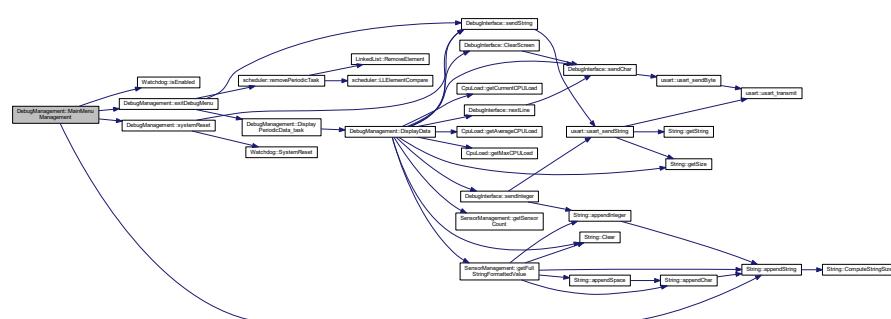
in	<i>rcv_char</i>	Character received on USART bus.
----	-----------------	----------------------------------

Returns

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

Definition at line 347 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.5.3.9 setInfoStringPtr()

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

Info message setting function.

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

Parameters

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

Returns

Nothing

Definition at line 126 of file DebugManagement.h.

4.5.3.10 systemReset()

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

System reset function.

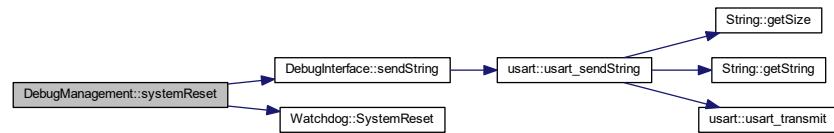
This function provokes a reset of the system. It displays a message on the screen and calls the reset function from watchdog class.

Returns

Nothing.

Definition at line 236 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.5.3.11 WatchdogMenuManagement()**

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

[Watchdog](#) menu management function.

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

Parameters

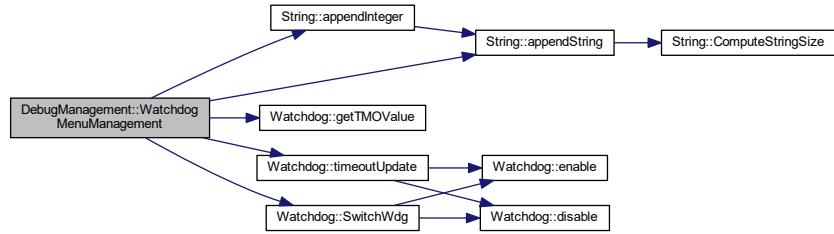
in	<code>rcv_char</code>	Character received on USART bus.
----	-----------------------	----------------------------------

Returns

Nothing.

Definition at line 242 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.5.4 Member Data Documentation

4.5.4.1 debug_ift_ptr

`DebugInterface* DebugManagement::debug_ift_ptr [private]`

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

Definition at line 133 of file DebugManagement.h.

4.5.4.2 debug_state

`debug_mgt_state_struct_t DebugManagement::debug_state [private]`

Structure containing debug states for each menu

Definition at line 137 of file DebugManagement.h.

4.5.4.3 info_string_ptr

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

Pointer to the info message to display

Definition at line 136 of file DebugManagement.h.

4.5.4.4 isInfoStringDisplayed

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

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

Definition at line 138 of file DebugManagement.h.

4.5.4.5 menu_string_ptr

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

Pointer to the current menu string to display

Definition at line 135 of file DebugManagement.h.

4.5.4.6 sensorMgt_ptr

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

Pointer to the sensor management object

Definition at line 134 of file DebugManagement.h.

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

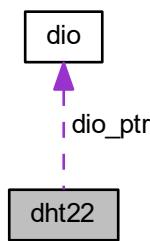
- [DebugManagement.h](#)
- [DebugManagement.cpp](#)

4.6 dht22 Class Reference

DHT 22 driver class.

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

Collaboration diagram for dht22:



Public Member Functions

- `dht22 (uint8_t port)`
dht22 class constructor.
- `bool getTemperature (uint16_t *temperature)`
Temperature get function.
- `bool getHumidity (uint16_t *humidity)`
Humidity get function.

Private Member Functions

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

Private Attributes

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

4.6.1 Detailed Description

DHT 22 driver class.

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

Definition at line 19 of file dht22.h.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 dht22()

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

[dht22](#) class constructor.

Initializes the class [dht22](#).

Parameters

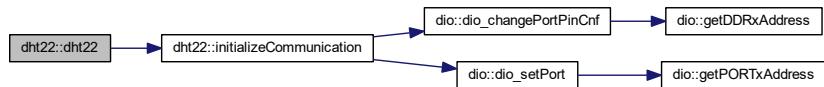
in	<i>port</i>	Encoded configuration of the port used for 1-wire communication.
----	-------------	--

Returns

Nothing

Definition at line 24 of file dht22.cpp.

Here is the call graph for this function:



4.6.3 Member Function Documentation

4.6.3.1 getHumidity()

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

Humidity get function.

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

Parameters

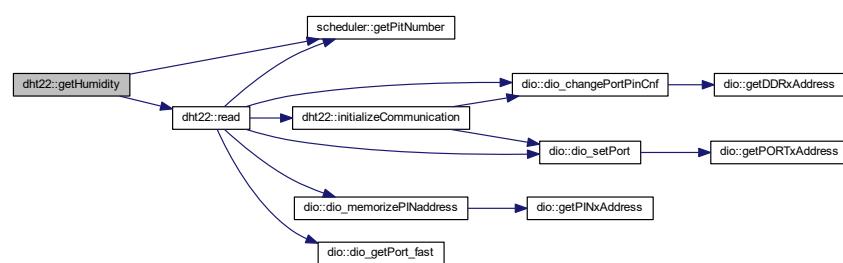
in	<i>humidity</i>	Address where the humidity shall be written
----	-----------------	---

Returns

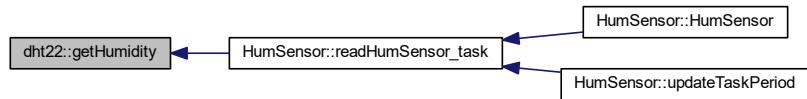
Validity of the data

Definition at line 220 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.3.2 getTemperature()

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

Temperature get function.

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

Parameters

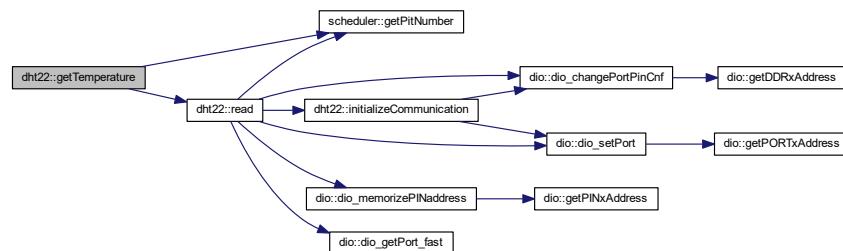
in	<i>temperature</i>	Address where the temperature shall be written
----	--------------------	--

Returns

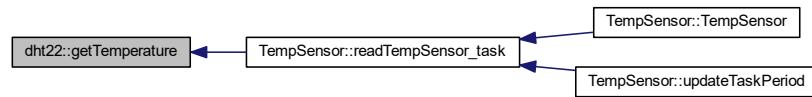
Validity of the data

Definition at line 231 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.3.3 initializeCommunication()

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

Initializes the communication.

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

Returns

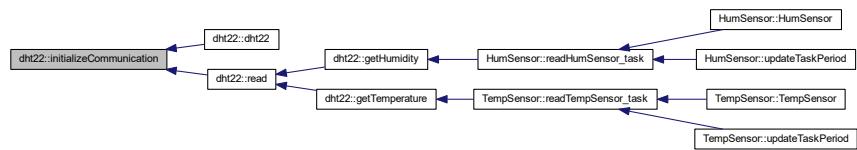
Nothing

Definition at line 210 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.3.4 read()

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

Reads the data from DHT22.

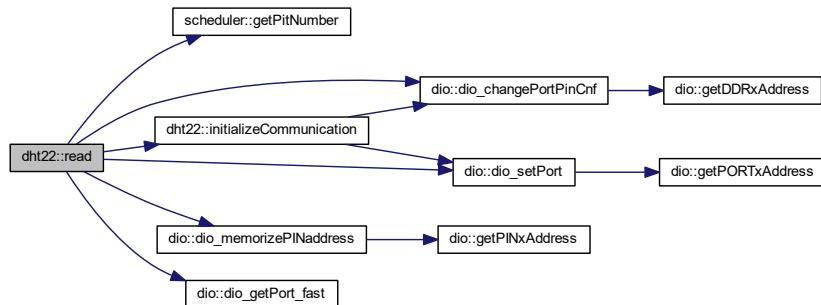
This function communicates with DHT22 using 1-wire protocol to read raw values of temperature and humidity. A checksum check is done when communication is finished to validate the received data. Validity of the data, temperature and humidity values are memorized in the associated class members.

Returns

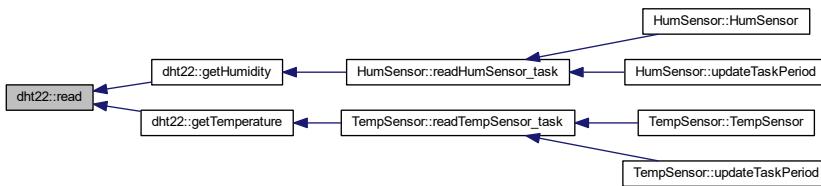
Nothing

Definition at line 34 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.4 Member Data Documentation

4.6.4.1 dht22_port

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

Variable containing the port used for 1-wire communication

Definition at line 55 of file dht22.h.

4.6.4.2 dio_ptr

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

Pointer to the DIO object

Definition at line 56 of file dht22.h.

4.6.4.3 mem_humidity

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

Memorized value of humidity

Definition at line 58 of file dht22.h.

4.6.4.4 mem_temperature

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

Memorized value of temperature

Definition at line 57 of file dht22.h.

4.6.4.5 mem_validity

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

Memorized value of validity

Definition at line 59 of file dht22.h.

4.6.4.6 pit_last_read

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

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

Definition at line 60 of file dht22.h.

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

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

4.7 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`

4.7.1 Detailed Description

DIO class.

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

Definition at line 27 of file dio.h.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 dio()

`dio::dio ()`

`dio` class constructor

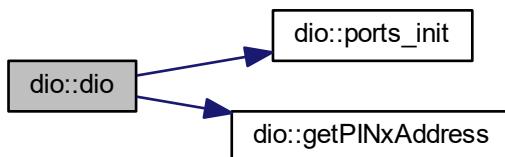
Initializes class `dio` and calls DIO hardware initialization function

Returns

Nothing

Definition at line 112 of file dio.cpp.

Here is the call graph for this function:



4.7.3 Member Function Documentation

4.7.3.1 dio_changePortPinCnf()

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

Changes the IO configuration of the selected pin.

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

Parameters

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

Returns

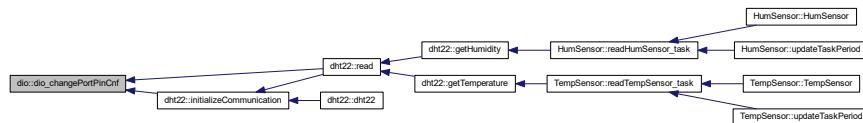
Nothing

Definition at line 149 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.7.3.2 dio_getPort()**

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

Gets the logical state of selected pin.

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

Parameters

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

Returns

Logical state of selected pin

Definition at line 139 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.7.3.3 dio_getPort_fast()**

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

Gets the logical state of the memorized pin.

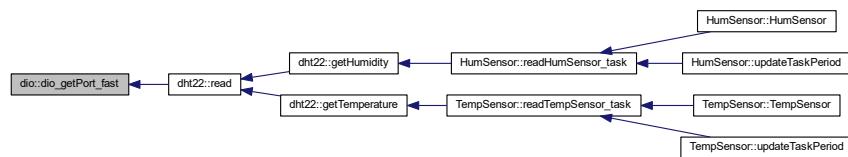
This function gets the logical value of the memorized pin. The corresponding port and pin index are stored in class members PINx_addr_mem and PINx_idx_mem. This mechanism is used to speed up reading time as this function no longer needs to extract register address and pin index from portcode.

Returns

Logical state of selected pin

Definition at line 171 of file dio.cpp.

Here is the caller graph for this function:



4.7.3.4 dio_invertPort()

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

Inverts the state of output port.

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

Parameters

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

Returns

Nothing

Definition at line 131 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.3.5 dio_memorizePINaddress()

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

Memorizes PINx register address and pin index.

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

Parameters

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

Returns

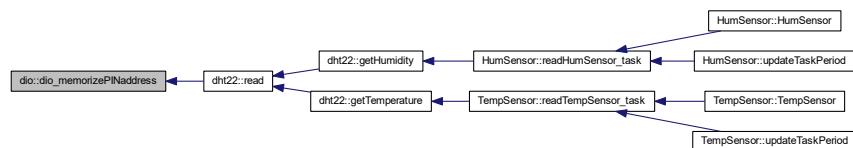
Nothing

Definition at line 165 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.3.6 dio_setPort()

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

Port setting function.

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

Parameters

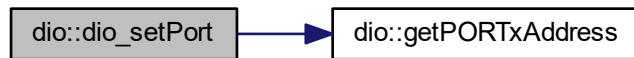
in	<i>portcode</i>	Encoded pin and register index
in	<i>state</i>	Requested state to set pin

Returns

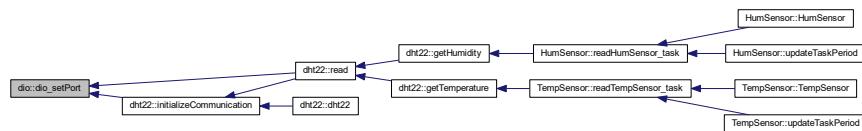
Nothing

Definition at line 121 of file dio.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.7.3.7 getDDRxAddress()**

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

Gets the physical address of the requested register DDRx.

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

Parameters

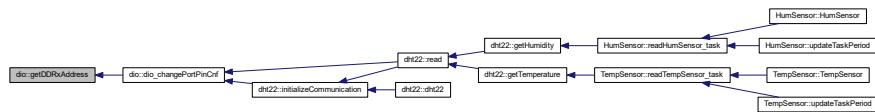
in	<i>portcode</i>	Encoded port code
----	-----------------	-------------------

Returns

Pointer to the DDRx register

Definition at line 83 of file dio.cpp.

Here is the caller graph for this function:



4.7.3.8 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 PIN x where x is encoded into the parameter portcode.

Parameters

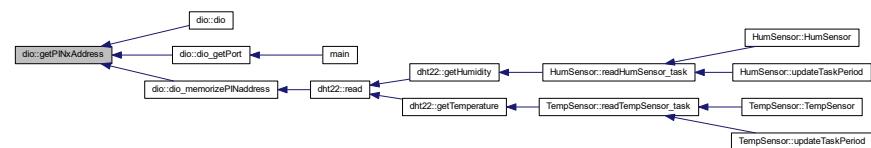
in	<i>portcode</i>	Encoded port code
----	-----------------	-------------------

Returns

Pointer to the PINx register

Definition at line 54 of file dio.cpp.

Here is the caller graph for this function:



4.7.3.9 getPORTxAddress()

```
uint8_t * dio::getPORTxAddress (
```

Gets the physical address of the requested register PORTx.

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

Parameters

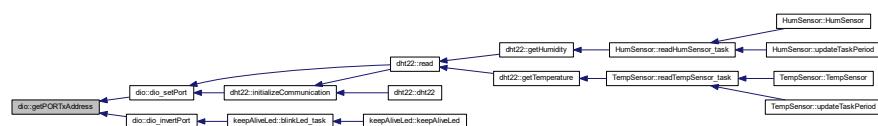
in	<i>portcode</i>	Encoded port code
----	-----------------	-------------------

Returns

Pointer to the PORTx register

Definition at line 25 of file dio.cpp.

Here is the caller graph for this function:

**4.7.3.10 ports_init()**

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

Digital ports hardware initialization function.

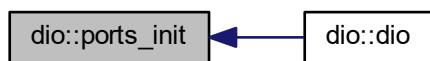
This function initializes digital ports as input or output and sets their initial values

Returns

Nothing

Definition at line 16 of file dio.cpp.

Here is the caller graph for this function:

**4.7.4 Member Data Documentation**

4.7.4.1 PINx_addr_mem

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

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

Definition at line 142 of file dio.h.

4.7.4.2 PINx_idx_mem

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

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

Definition at line 143 of file dio.h.

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

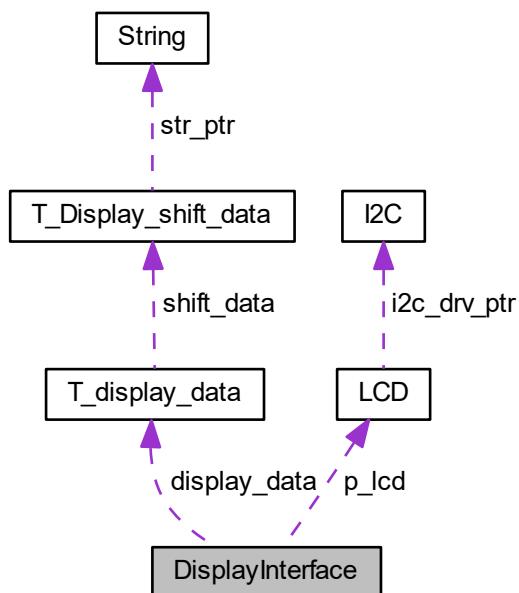
- [dio.h](#)
- [dio.cpp](#)

4.8 DisplayInterface Class Reference

Display interface services class.

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

Collaboration diagram for DisplayInterface:



Public Member Functions

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

Static Public Member Functions

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

Private Member Functions

- `uint8_t FindFirstCharAddr (uint8_t line)`
Finds start address of a line.
- `void RefreshLine (uint8_t line)`
Line refresh function.
- `void ClearStringInDataStruct (uint8_t line)`
String data clearing structure.
- `void setLineAlignment (uint8_t line)`
Text alignment setting function.

Private Attributes

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

4.8.1 Detailed Description

Display interface services class.

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

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

4.8.2 Constructor & Destructor Documentation

4.8.2.1 `DisplayInterface()`

```
DisplayInterface::DisplayInterface (
    const T\_LCD\_conf\_struct * LCD_init_cnf )
```

Class constructor.

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

Parameters

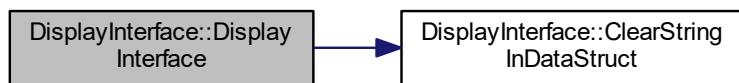
in	<i>LCD_init_cnf</i>	Initial configuration of the screen
----	---------------------	-------------------------------------

Returns

Nothing

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

Here is the call graph for this function:



4.8.3 Member Function Documentation

4.8.3.1 ClearFullScreen()

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

Screen cleaning function.

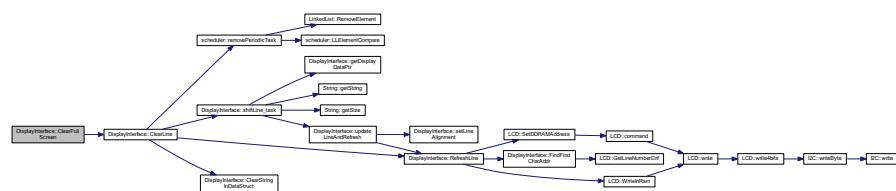
This functions clears the entire display. It uses the ClearLine function on every line of the screen.

Returns

Nothing

Definition at line 269 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.3.2 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

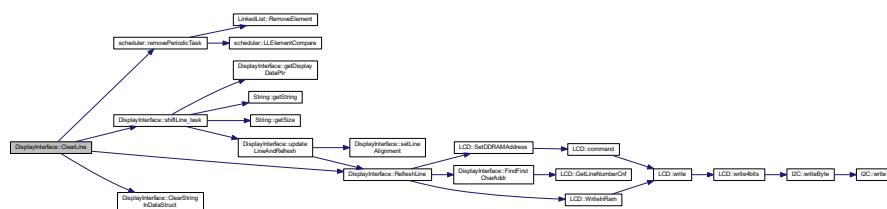
in	<i>line</i>	Line to clear
----	-------------	---------------

Returns

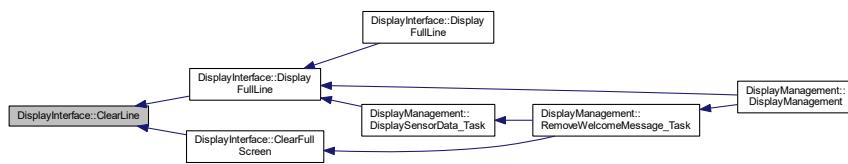
True if the line has been cleared, false otherwise

Definition at line 224 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.3.3 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

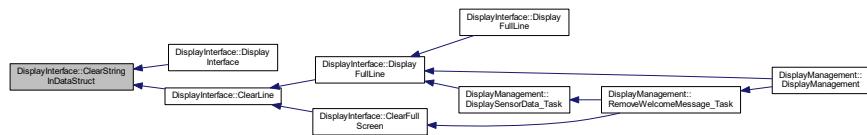
in	line	Line to clear
----	------	---------------

Returns

Nothing

Definition at line 177 of file DisplayInterface.cpp.

Here is the caller graph for this function:



4.8.3.4 DisplayFullLine() [1/2]

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

Line display function.

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

Parameters

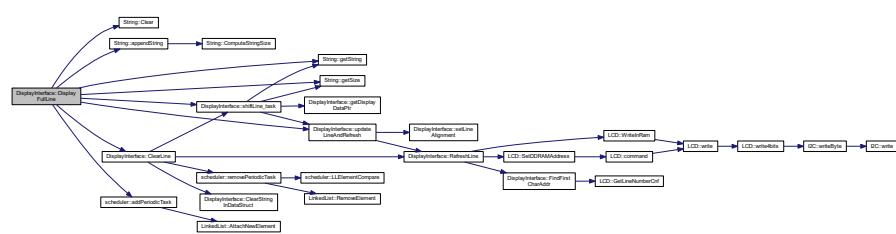
in	<i>str</i>	Pointer to the string to display
in	<i>size</i>	Size of the string to display
in	<i>line</i>	Index of the line where the string shall be displayed
in	<i>mode</i>	Display mode
in	<i>alignment</i>	Requested alignment for the line

Returns

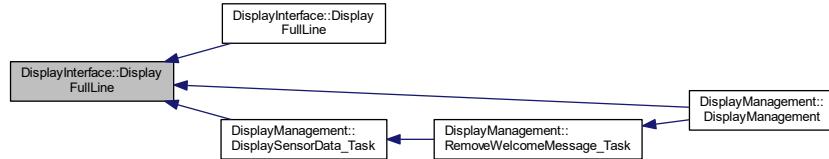
True if the line has been correctly displayed, false otherwise

Definition at line 59 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.3.5 DisplayFullLine() [2/2]

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

Line display function.

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

Parameters

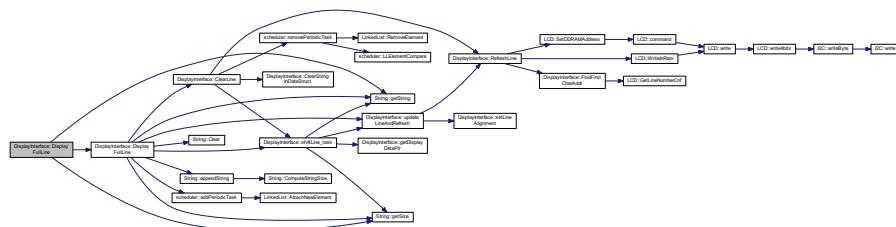
in	<i>str</i>	Pointer to the string to display
in	<i>line</i>	Index of the line where the string shall be displayed
in	<i>mode</i>	Display mode
in	<i>alignment</i>	Requested alignment for the line

Returns

True if the line has been correctly displayed, false otherwise

Definition at line 143 of file DisplayInterface.cpp.

Here is the call graph for this function:



4.8.3.6 FindFirstCharAddr()

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

Finds start address of a line.

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

Parameters

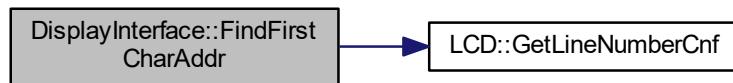
<code>in</code>	<code>line</code>	Line which address shall be found
-----------------	-------------------	-----------------------------------

Returns

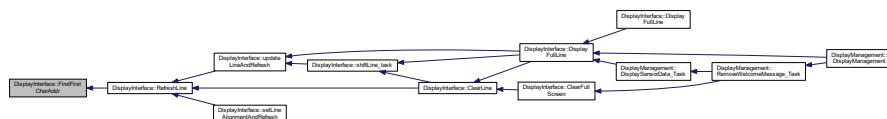
Address in DDRAM of the first character of the line

Definition at line 185 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.3.7 getDisplayDataPtr()

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

Display data get function.

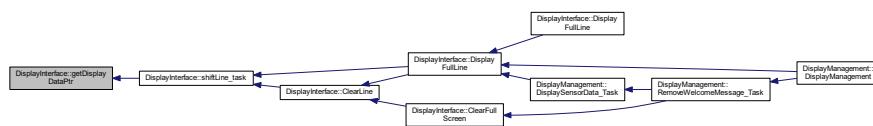
This function returns a pointer to the display data structure.

Returns

Pointer to display data structure.

Definition at line 154 of file DisplayInterface.h.

Here is the caller graph for this function:

**4.8.3.8 IsLineEmpty()**

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

Empty line get function.

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

Parameters

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

Returns

True if the line is empty, false otherwise

Definition at line 277 of file DisplayInterface.cpp.

4.8.3.9 RefreshLine()

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

Line refresh function.

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

Parameters

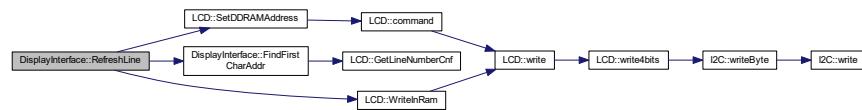
in	<i>line</i>	Line to refresh
----	-------------	-----------------

Returns

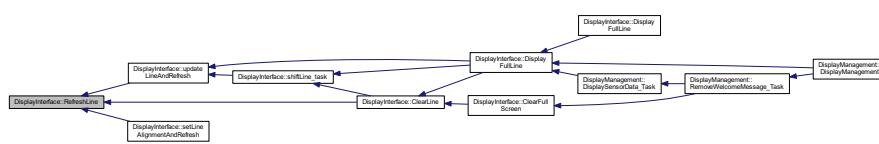
Nothing

Definition at line 164 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.8.3.10 setLineAlignment()**

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

Text alignment setting function.

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

Parameters

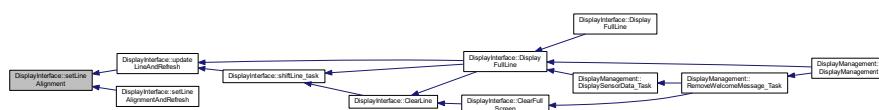
in	<i>line</i>	Line to update
----	-------------	----------------

Returns

Nothing

Definition at line 334 of file DisplayInterface.cpp.

Here is the caller graph for this function:



4.8.3.11 setLineAlignmentAndRefresh()

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

Text alignment function.

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

Parameters

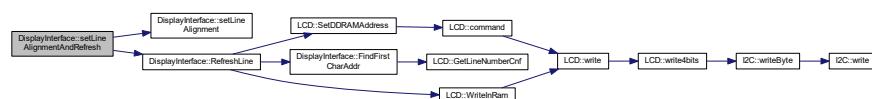
in	<i>line</i>	Requested line to update
in	<i>alignment</i>	Requested alignment for the text

Returns

Nothing

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

Here is the call graph for this function:



4.8.3.12 shiftLine_task()

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

Line shifting periodic task.

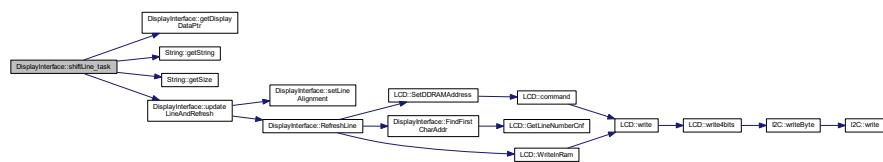
This function is called periodically by the scheduler. It shifts all the lines in line shifting mode and updates the data structures.

Returns

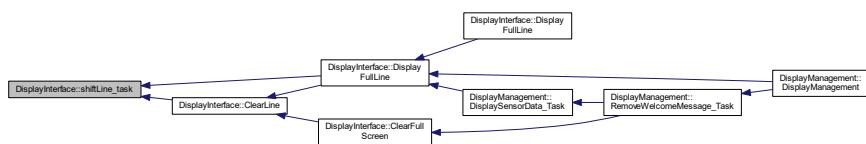
Nothing

Definition at line 286 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.8.3.13 updateLineAndRefresh()**

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

Line data string update function.

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

Parameters

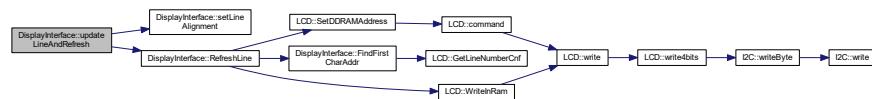
in	str	Pointer to the string to display
in	size	Size of the string
in	line	Line to update

Returns

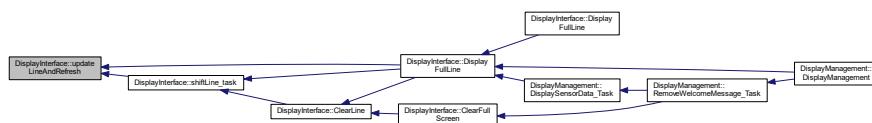
Nothing

Definition at line 148 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.4 Member Data Documentation

4.8.4.1 display_data

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

Screen display data

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

4.8.4.2 dummy

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

Needed for data alignment

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

4.8.4.3 isShiftInProgress

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

Flag indicating if a shift is in progress on any line

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

4.8.4.4 p_lcd

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

Pointer to the attached LCD driver object

Definition at line 186 of file DisplayInterface.h.

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

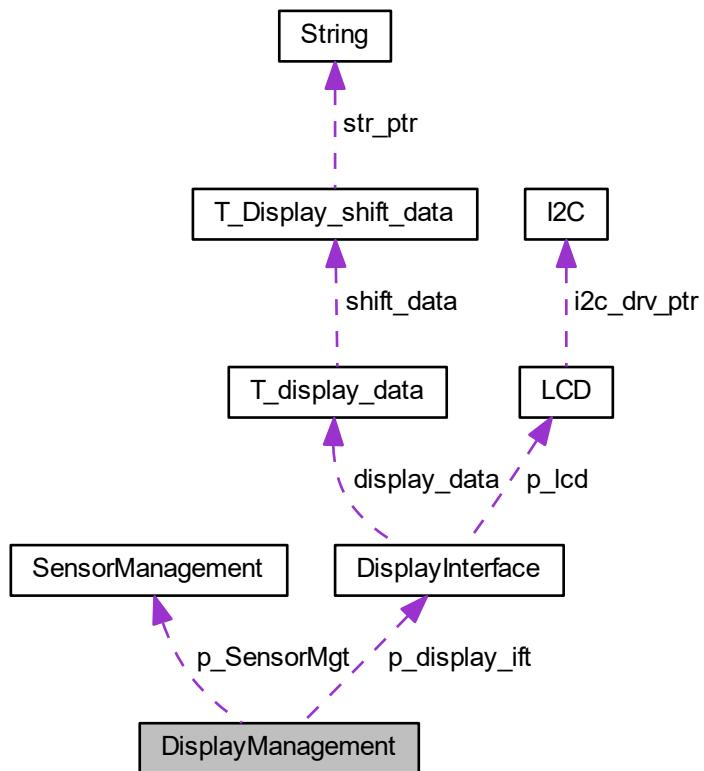
- [DisplayInterface.h](#)
- [DisplayInterface.cpp](#)

4.9 DisplayManagement Class Reference

Display management class.

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

Collaboration diagram for DisplayManagement:



Public Member Functions

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

Static Public Member Functions

- static void **DisplaySensorData_Task** ()
Periodic task for displaying sensor data.
 - static void **RemoveWelcomeMessage_Task** ()
End of welcome message task.

Private Attributes

- `DisplayInterface * p_display_ift`
 - `SensorManagement * p_SensorMgt`

4.9.1 Detailed Description

Display management class.

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

Definition at line 47 of file DisplayManagement.h.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 DisplayManagement()

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

Class constructor.

This class initializes display management.

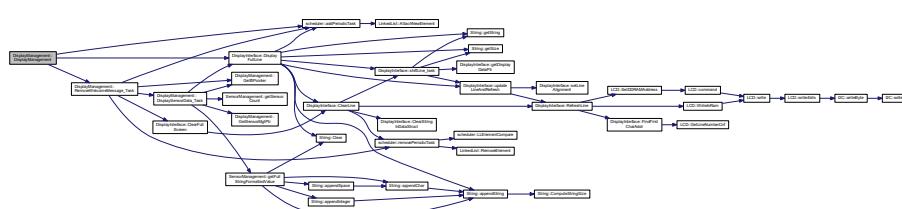
It created a display interface object and initializes all class variables.

Returns

Nothing

Definition at line 30 of file DisplayManagement.cpp.

Here is the call graph for this function:



4.9.3 Member Function Documentation

4.9.3.1 DisplaySensorData_Task()

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

Periodic task for displaying sensor data.

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

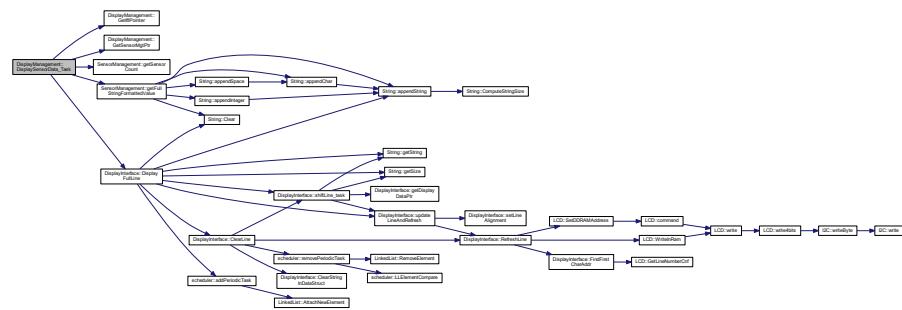
It is called periodically by scheduler.

Returns

Nothing

Definition at line 71 of file DisplayManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.3.2 GetIfPointer()

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

Interface pointer get function.

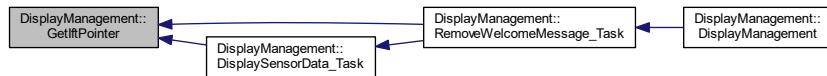
This function returns the pointer to the display interface object

Returns

Pointer to display interface object

Definition at line 76 of file DisplayManagement.h.

Here is the caller graph for this function:



4.9.3.3 GetSensorMgtPtr()

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

Sensor management pointer get function.

This function returns the pointer to the sensor management object

Returns

Pointer to sensor management object

Definition at line 87 of file DisplayManagement.h.

Here is the caller graph for this function:



4.9.3.4 RemoveWelcomeMessage_Task()

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

End of welcome message task.

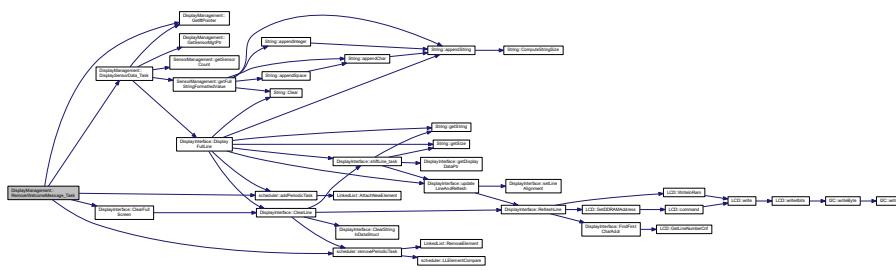
This task clears the welcome message from the screen and start periodic display of sensor data. This task shall be added in scheduler when the welcome message is displayed on screen. As it shall be called only once, the task removes itself from the scheduler after the first call.

Returns

Nothing

Definition at line 54 of file DisplayManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.9.4 Member Data Documentation

4.9.4.1 p_display_ift

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

Pointer to the display interface object

Definition at line 104 of file DisplayManagement.h.

4.9.4.2 p_SensorMgt

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

Pointer to the sensor management object

Definition at line 105 of file DisplayManagement.h.

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

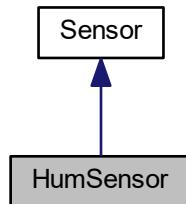
- [DisplayManagement.h](#)
- [DisplayManagement.cpp](#)

4.10 HumSensor Class Reference

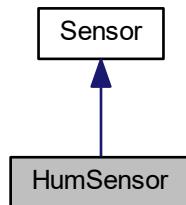
Class for humidity sensor.

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

Inheritance diagram for HumSensor:



Collaboration diagram for HumSensor:



Public Member Functions

- **HumSensor ()**
Class constructor.
 - **HumSensor (uint16_t val_tmo, uint16_t period)**
Overloaded class constructor.
 - **bool updateTaskPeriod (uint16_t period)**
Task period update.

Static Public Member Functions

- static void `readHumSensor_task ()`
Task for reading humidity values.

Additional Inherited Members

4.10.1 Detailed Description

Class for humidity sensor.

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

Definition at line 18 of file HumSensor.h.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 HumSensor() [1/2]

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

Class constructor.

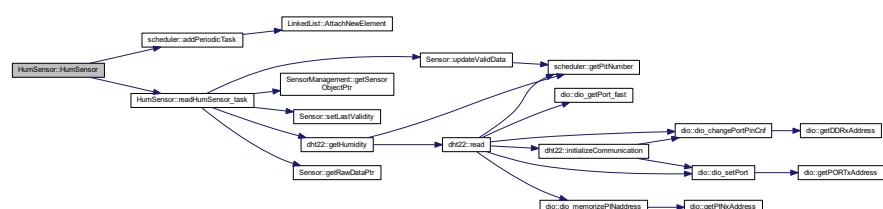
This function initializes all data of the class `HumSensor`. If needed, it creates a new instance of the DHT22 sensor object. It also adds the periodic task in the scheduler.

Returns

Nothing

Definition at line 28 of file HumSensor.cpp.

Here is the call graph for this function:



4.10.2.2 HumSensor() [2/2]

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

Overloaded class constructor.

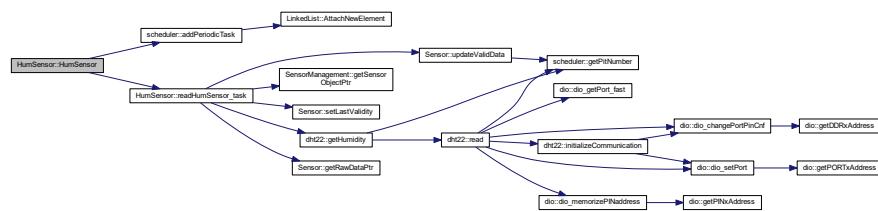
This function initializes all data of the class `HumSensor`. It sets validity timeout and task period to the given value. If needed, it creates a new instance of the DHT22 sensor object. It also adds the periodic task in the scheduler.

Returns

Nothing

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

Here is the call graph for this function:



4.10.3 Member Function Documentation

4.10.3.1 readHumSensor_task()

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

Task for reading humidity values.

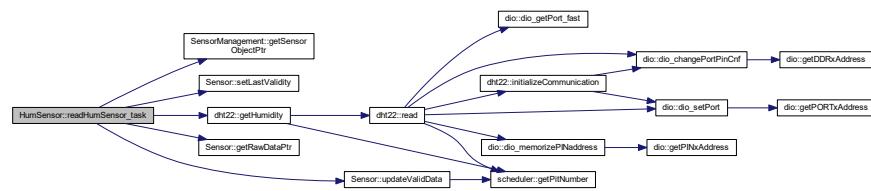
This task reads humidity data using DHT22 driver. It is called periodically.

Returns

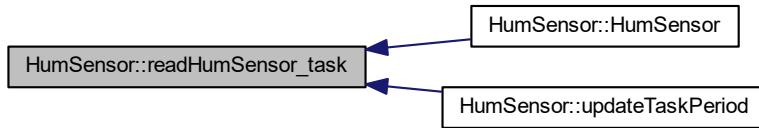
Nothing

Definition at line 48 of file HumSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.10.3.2 updateTaskPeriod()**

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

Task period update.

This function updates the period of the temperature task.

Parameters

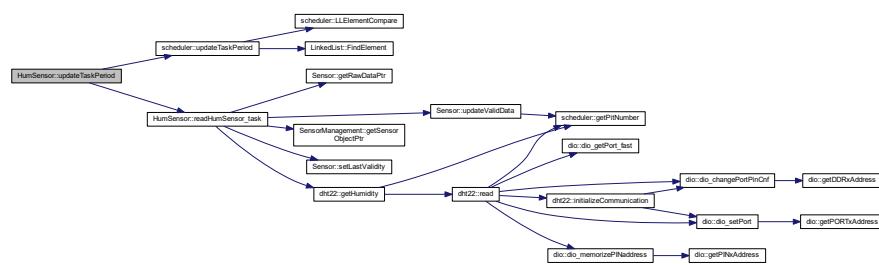
in	<i>period</i>	New period of the task
----	---------------	------------------------

Returns

True if the period has been updated, false otherwise

Definition at line 62 of file HumSensor.cpp.

Here is the call graph for this function:



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

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

4.11 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 \(uint8_t data, uint8_t tx_address, bool sendStopCond\)`](#)
Byte sending function.
- [`bool write \(uint8_t *data, uint8_t tx_address, uint8_t size, bool sendStopCond\)`](#)
I2C write function.
- [`bool read \(uint8_t i2c_address, uint8_t size, uint8_t *buf_ptr\)`](#)
I2C read function.
- [`void setBitRate \(uint32_t l_bitrate\)`](#)
Variable bitrate setting function.

Private Member Functions

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

Private Attributes

- `uint32_t bitrate`

4.11.1 Detailed Description

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

This class manages [I2C](#) driver.

Definition at line 25 of file I2C.h.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 I2C()

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

[I2C](#) class constructor.

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

Parameters

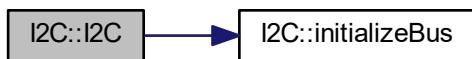
in	<i>l_bitrate</i>	Requested bitrate for I2C bus (in Hz)
----	------------------	---

Returns

Nothing

Definition at line 16 of file I2C.cpp.

Here is the call graph for this function:



4.11.3 Member Function Documentation

4.11.3.1 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 129 of file I2C.cpp.

Here is the caller graph for this function:



4.11.3.2 read()

```
bool I2C::read (
    uint8_t i2c_address,
    uint8_t size,
    uint8_t * buf_ptr )
```

I2C read function.

This function performs a read operation on the I2C. The requested number of bytes is read on the bus and the received data are copied in the reception buffer. Calling function has to check that enough space is allocated for the reception buffer.

Parameters

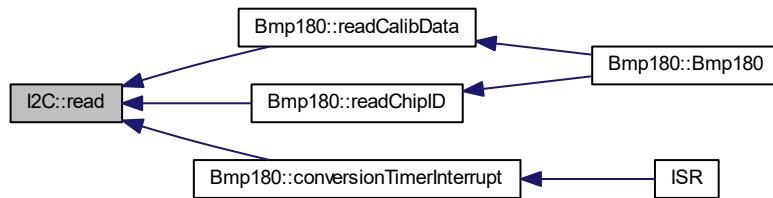
in	<i>i2c_address</i>	I2C address of the device
in	<i>size</i>	Number of bytes to read
out	<i>buf_ptr</i>	Pointer to the start of the reception buffer.

Returns

True if the receive process has succeeded, false otherwise

Definition at line 75 of file I2C.cpp.

Here is the caller graph for this function:

**4.11.3.3 setBitRate()**

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

Variable bitrate setting function.

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

Parameters

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

Returns

Nothing

Definition at line 124 of file I2C.cpp.

4.11.3.4 write()

```
bool I2C::write (
    uint8_t * data,
    uint8_t tx_address,
    uint8_t size,
    bool sendStopCond )
```

I2C write function.

This function sends the requested number of bytes to the I2C device with the given address

Parameters

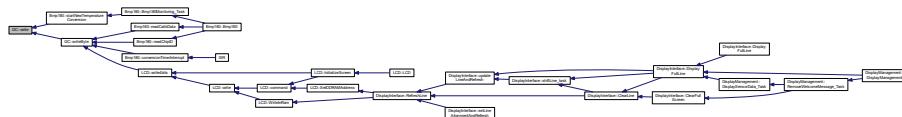
in	<i>data</i>	Pointer to the data to send
in	<i>tx_address</i>	I2C address of the device
in	<i>size</i>	Number of bytes to send
in	<i>sendStopCond</i>	Defines if the stop condition shall be sent or not

Returns

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

Definition at line 28 of file I2C.cpp.

Here is the caller graph for this function:

**4.11.3.5 writeByte()**

```
bool I2C::writeByte (
    uint8_t data,
    uint8_t tx_address,
    bool sendStopCond )
```

Byte sending function.

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

Parameters

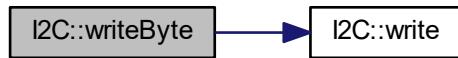
in	<i>data</i>	Data to send
in	<i>tx_address</i>	I2C address of the device
in	<i>sendStopCond</i>	Defines if the stop condition shall be sent or not

Returns

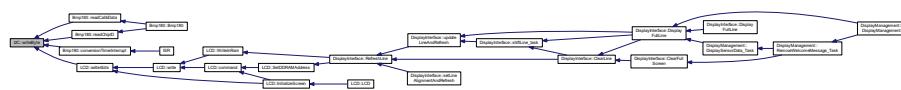
True if transmission is completed, False if an error has occurred

Definition at line 23 of file I2C.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.11.4 Member Data Documentation

4.11.4.1 bitrate

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

Definition at line 83 of file I2C.h.

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

- I2C.h
 - I2C.cpp

4.12 keepAliveLed Class Reference

Class for keep-alive LED blinking.

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

Public Member Functions

- `keepAlive() ed`

Class constructor

Static Public Member Functions

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

Task for LED blinking.

4.12.1 Detailed Description

Class for keep-alive LED blinking.

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

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

4.12.2 Constructor & Destructor Documentation

4.12.2.1 `keepAliveLed()`

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

Class constructor.

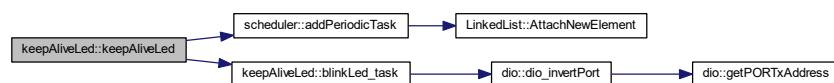
This function initializes the class `keepAliveLed`

Returns

Nothing

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

Here is the call graph for this function:



4.12.3 Member Function Documentation

4.12.3.1 blinkLed_task()

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

Task for LED blinking.

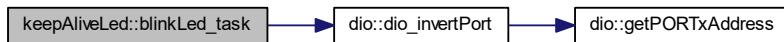
This function is inserted into the scheduler. It changes the state of the LED output to make it blink

Returns

Nothing

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

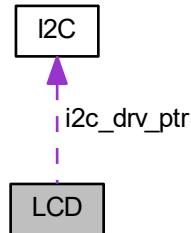
- [keepAliveLed.h](#)
- [keepAliveLed.cpp](#)

4.13 LCD Class Reference

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

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

Collaboration diagram for LCD:



Public Member Functions

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

Private Member Functions

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

Private Attributes

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

4.13.1 Detailed Description

Class for `LCD` S2004A display driver.

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

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

4.13.2 Constructor & Destructor Documentation

4.13.2.1 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

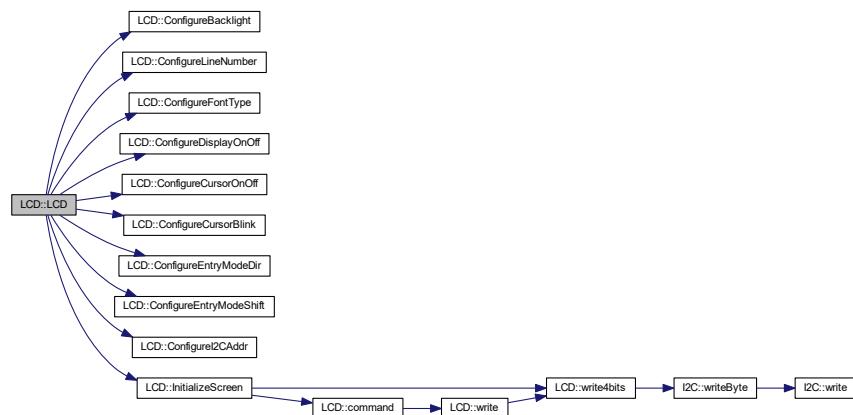
in	<code>init_conf</code>	Initial configuration structure
----	------------------------	---------------------------------

Returns

Nothing

Definition at line 19 of file LCD.cpp.

Here is the call graph for this function:



4.13.3 Member Function Documentation

4.13.3.1 command()

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

LCD command management function.

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

Parameters

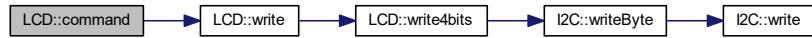
in	<i>cmd</i>	Requested command
----	------------	-------------------

Returns

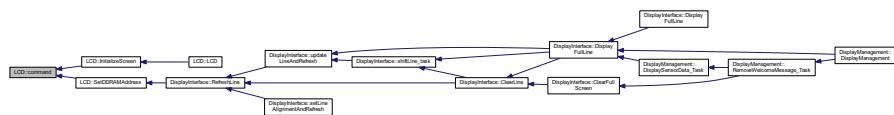
Nothing

Definition at line 129 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.13.3.2 ConfigureBacklight()

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

Backlight configuration function.

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

Parameters

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

Returns

Nothing

Definition at line 178 of file LCD.h.

Here is the caller graph for this function:



4.13.3.3 ConfigureCursorBlink()

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

Cursor blinking configuration function.

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

Parameters

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

Returns

Nothing

Definition at line 238 of file LCD.h.

Here is the caller graph for this function:



4.13.3.4 ConfigureCursorOnOff()

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

Cursor configuration function.

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

Parameters

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

Returns

Nothing

Definition at line 226 of file LCD.h.

Here is the caller graph for this function:



4.13.3.5 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:



4.13.3.6 ConfigureEntryModeDir()

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

Entry mode direction configuration function.

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

Parameters

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

Returns

Nothing

Definition at line 250 of file LCD.h.

Here is the caller graph for this function:

**4.13.3.7 ConfigureEntryModeShift()**

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

Entry mode shift configuration function.

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

Parameters

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

Returns

Nothing

Definition at line 262 of file LCD.h.

Here is the caller graph for this function:



4.13.3.8 ConfigureFontType()

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

Font configuration function.

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

Parameters

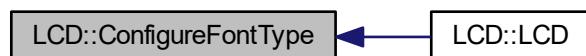
in	param	Configuration value
----	-------	---------------------

Returns

Nothing

Definition at line 202 of file LCD.h.

Here is the caller graph for this function:



4.13.3.9 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

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

Returns

Nothing

Definition at line 274 of file LCD.h.

Here is the caller graph for this function:



4.13.3.10 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

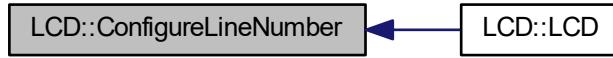
<code>in</code>	<code>param</code>	Configuration value
-----------------	--------------------	---------------------

Returns

Nothing

Definition at line 190 of file LCD.h.

Here is the caller graph for this function:



4.13.3.11 GetDDRAMAddress()

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

DDRAM address get function.

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

Returns

Current DDRAM address

Definition at line 294 of file LCD.h.

4.13.3.12 GetLineNumberCnf()

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

Number of line get function.

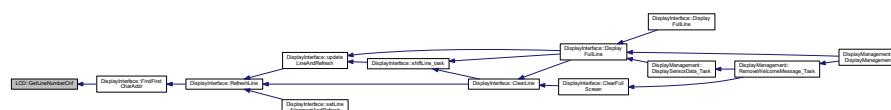
This function returns the line number configuration of the screen : 1 or 2 lines mode.

Returns

Line number configuration

Definition at line 316 of file LCD.h.

Here is the caller graph for this function:



4.13.3.13 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 79 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.13.3.14 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

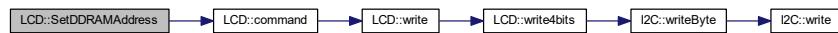
in	<i>addr</i>	New DDRAM address
----	-------------	-------------------

Returns

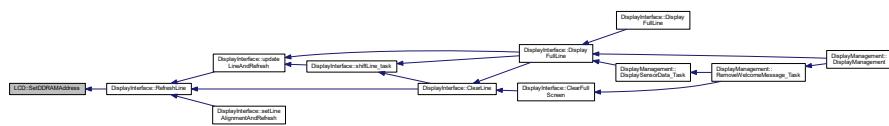
Nothing

Definition at line 172 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.13.3.15 write()**

```

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

[I2C](#) write function.

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

Parameters

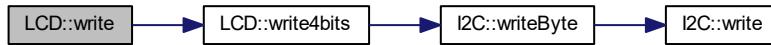
in	<i>data</i>	8-bit data for D0-7 pins of screen
in	<i>mode</i>	Requested mode for LCD communication

Returns

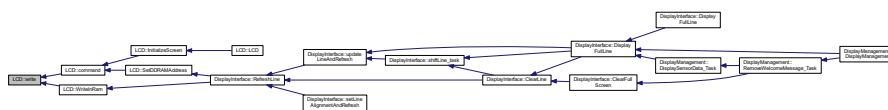
Nothing

Definition at line 63 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.13.3.16 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

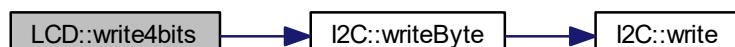
in	<i>data</i>	8-bit data to send
----	-------------	--------------------

Returns

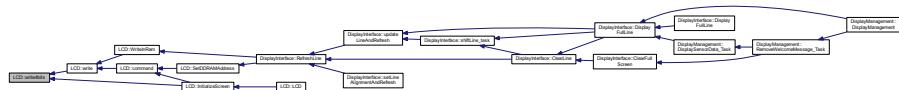
Nothing

Definition at line 46 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.13.3.17 WriteInRam()

```
void LCD::WriteInRam (
```

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

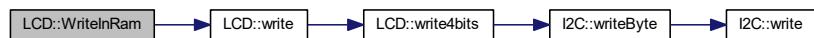
in	<i>a_char</i>	Data byte to write in RAM
in	<i>area</i>	Area in RAM where the data will be written : DDRAM or CGRAM

Returns

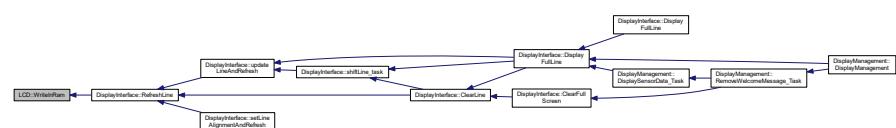
Nothing

Definition at line 194 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.13.4 Member Data Documentation

4.13.4.1 `backlight_enable`

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

Backlight enable flag

Definition at line 324 of file LCD.h.

4.13.4.2 `cnfCursorBlink`

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

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

Definition at line 329 of file LCD.h.

4.13.4.3 `cnfCursorOnOff`

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

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

Definition at line 328 of file LCD.h.

4.13.4.4 `cnfDisplayOnOff`

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

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

Definition at line 327 of file LCD.h.

4.13.4.5 `cnfEntryModeDir`

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

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

Definition at line 330 of file LCD.h.

4.13.4.6 cnfEntryModeShift

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

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

Definition at line 331 of file LCD.h.

4.13.4.7 cnfFontType

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

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

Definition at line 326 of file LCD.h.

4.13.4.8 cnfI2C_addr

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

I2C address of the [LCD](#) screen

Definition at line 332 of file LCD.h.

4.13.4.9 cnfLineNumber

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

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

Definition at line 325 of file LCD.h.

4.13.4.10 ddram_addr

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

Screen DDRAM address

Definition at line 336 of file LCD.h.

4.13.4.11 i2c_drv_ptr

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

Pointer to the [I2C](#) driver object

Definition at line 334 of file LCD.h.

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

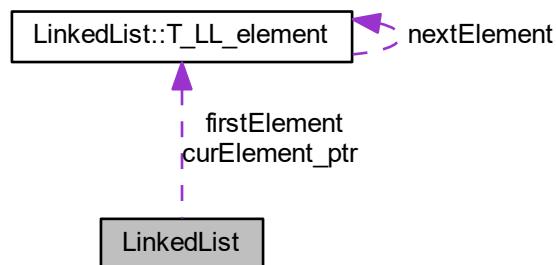
- [LCD.h](#)
- [LCD.cpp](#)

4.14 LinkedList Class Reference

Linked list class.

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

Collaboration diagram for [LinkedList](#):



Classes

- struct [T_LL_element](#)

Type defining a linked list element.

Public Member Functions

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

Private Types

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

Private Attributes

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

4.14.1 Detailed Description

Linked list class.

This class defines a linked list and the associated services.

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

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

4.14.2 Member Typedef Documentation

4.14.2.1 T_LL_element

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

Type defining a linked list element.

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

4.14.3 Constructor & Destructor Documentation

4.14.3.1 LinkedList()

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

Class constructor.

This constructor initializes a linked list

Returns

Nothing

Definition at line 18 of file LinkedList.cpp.

4.14.3.2 ~LinkedList()

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

Class destructor.

This function deletes the linked list

Returns

Nothing

Definition at line 24 of file LinkedList.cpp.

Here is the call graph for this function:



4.14.4 Member Function Documentation

4.14.4.1 AttachNewElement()

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

Add an new element to the list.

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

Parameters

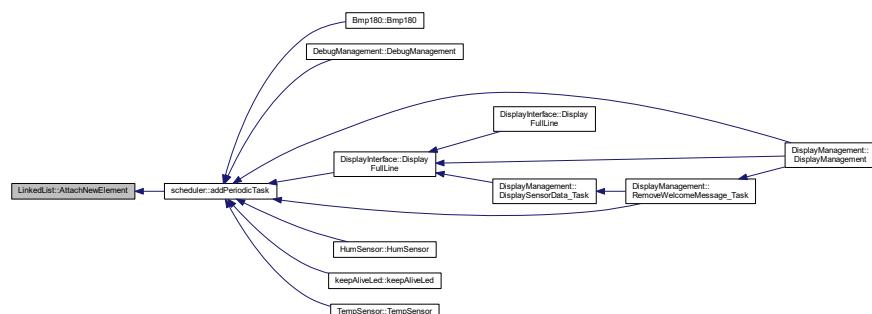
in	<i>data_ptr</i>	Pointer to the data element
----	-----------------	-----------------------------

Returns

Nothing

Definition at line 42 of file LinkedList.cpp.

Here is the caller graph for this function:



4.14.4.2 FindElement()

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

Element finding function.

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

Parameters

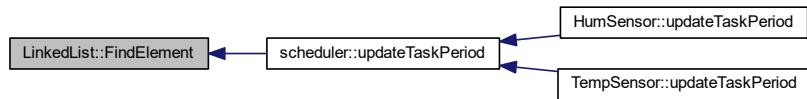
in	<i>comparisonFct_ptr</i>	Pointer to the comparison function
in	<i>reference_ptr</i>	Pointer to the element to find in the chain
out	<i>chainElement_ptr</i>	Pointer to pointer to the found element

Returns

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

Definition at line 133 of file LinkedList.cpp.

Here is the caller graph for this function:

**4.14.4.3 getCurrentElement()**

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

Current element get function.

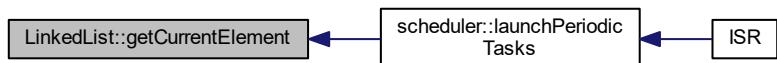
This function returns a pointer to the current pointed data in the chain.

Returns

Pointer to the current data

Definition at line 67 of file LinkedList.h.

Here is the caller graph for this function:



4.14.4.4 IsLLEmpty()

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

Empty linked list.

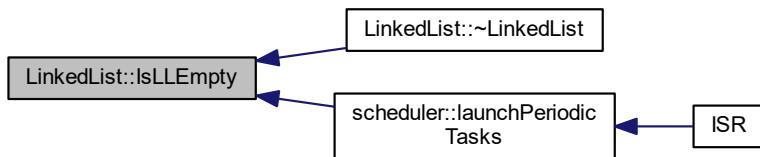
This function checks whether the linked list is empty or not (pointer to first element is equal to 0 or not).

Returns

True if the list is empty, false otherwise

Definition at line 125 of file LinkedList.cpp.

Here is the caller graph for this function:



4.14.4.5 MoveToNextElement()

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

Move to next element function.

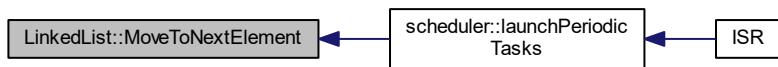
This function moves the element pointer to the next element of the chain.

Returns

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

Definition at line 111 of file LinkedList.cpp.

Here is the caller graph for this function:



4.14.4.6 RemoveElement()

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

Removes an element from the chain.

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

Parameters

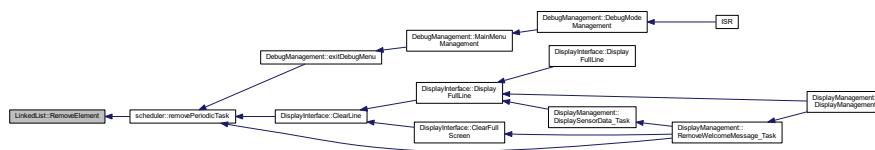
in	<i>comparisonFct_ptr</i>	Pointer to the comparison function to use
in	<i>reference_ptr</i>	Pointer to the reference data used for comparison

Returns

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

Definition at line 67 of file LinkedList.cpp.

Here is the caller graph for this function:



4.14.4.7 ResetElementPtr()

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

Resets element pointer.

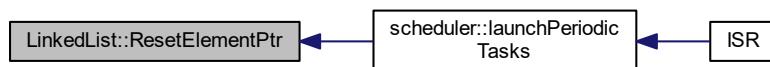
This function sets the element pointer to the first element of the chain.

Returns

Nothing

Definition at line 86 of file LinkedList.h.

Here is the caller graph for this function:



4.14.5 Member Data Documentation

4.14.5.1 curElement_ptr

`T_LL_element* LinkedList::curElement_ptr [private]`

Pointer to the current element of the list

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

4.14.5.2 firstElement

`T_LL_element* LinkedList::firstElement [private]`

Pointer to the first element of the list

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

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

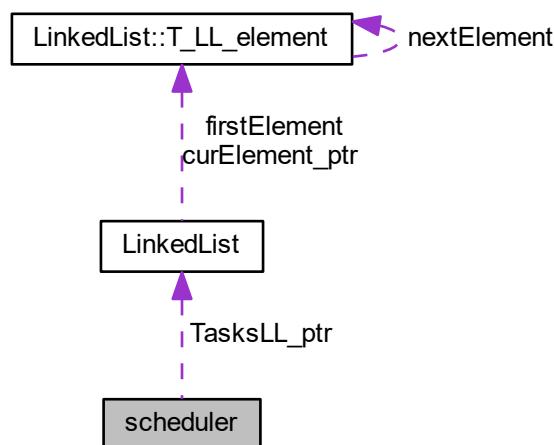
- [LinkedList.h](#)
- [LinkedList.cpp](#)

4.15 scheduler Class Reference

Scheduler class.

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

Collaboration diagram for scheduler:



Classes

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

Public Member Functions

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

Static Public Member Functions

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

Private Types

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

Private Attributes

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

4.15.1 Detailed Description

Scheduler class.

This class defines the scheduler of the system.

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

Definition at line 30 of file scheduler.h.

4.15.2 Member Typedef Documentation

4.15.2.1 Task_t

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

Type defining a task structure.

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

4.15.3 Constructor & Destructor Documentation

4.15.3.1 scheduler()

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

scheduler class constructor

This function initializes the class scheduler

Returns

Nothing

Definition at line 29 of file scheduler.cpp.

Here is the call graph for this function:



4.15.4 Member Function Documentation

4.15.4.1 addPeriodicTask()

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

Add a task into the scheduler.

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

Parameters

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

Returns

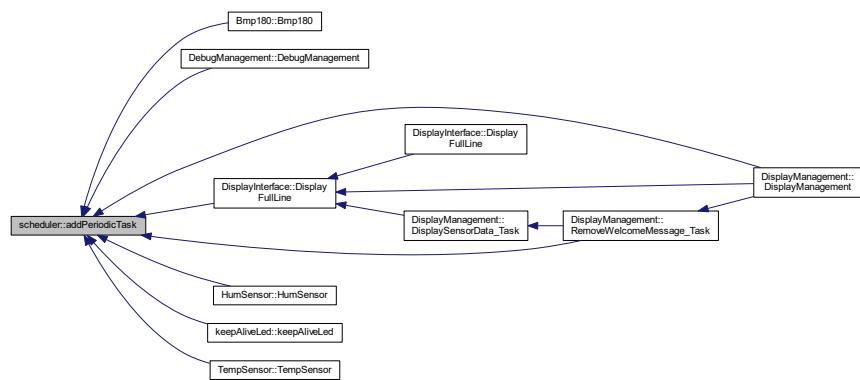
Nothing

Definition at line 99 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.15.4.2 getPitNumber()**

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

Get function for PIT number.

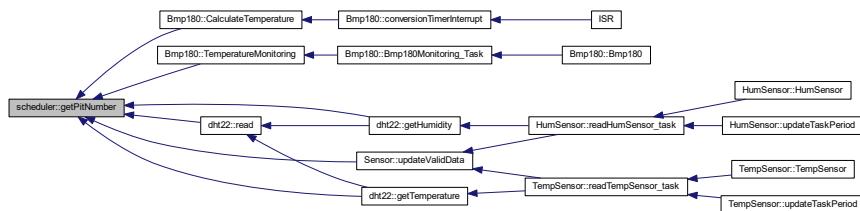
This function returns the PIT number

Returns

PIT number

Definition at line 113 of file scheduler.cpp.

Here is the caller graph for this function:

**4.15.4.3 getTaskCount()**

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

Task count get function.

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

Returns

Number of tasks

Definition at line 115 of file scheduler.h.

4.15.4.4 launchPeriodicTasks()

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

Main scheduler function.

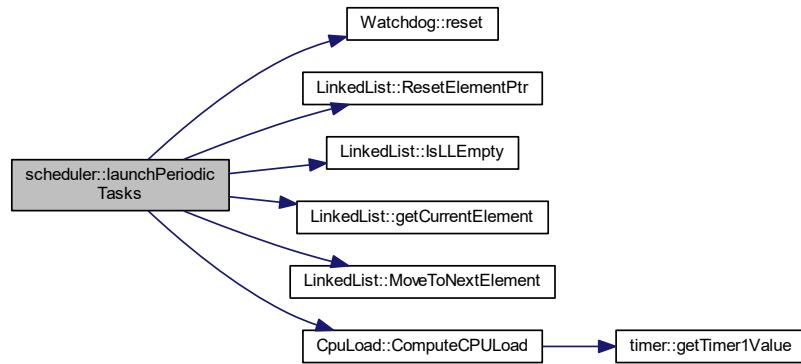
This function launches the scheduled tasks according to current software time and task configuration

Returns

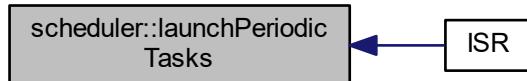
Nothing

Definition at line 54 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.15.4.5 LLElementCompare()

```

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

Linked list comparison function.

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

Parameters

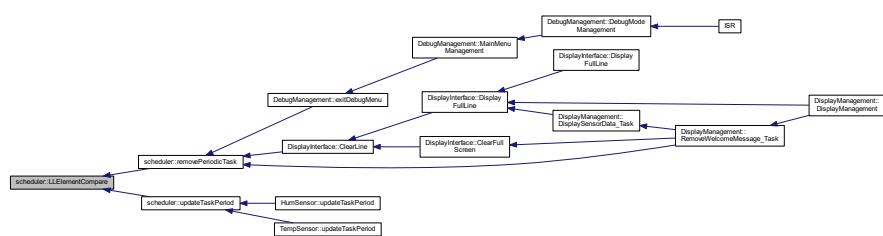
in	<i>LLElement</i>	Pointer to the linked list element
in	<i>CompareElement</i>	Pointer to the element to the compare

Returns

True if both elements are identical, false otherwise

Definition at line 131 of file scheduler.cpp.

Here is the caller graph for this function:

**4.15.4.6 removePeriodicTask()**

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

Remove a task from the scheduler.

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

Parameters

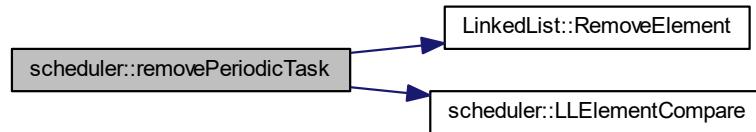
in	<i>task_ptr</i>	address of the task to remove from scheduler
----	-----------------	--

Returns

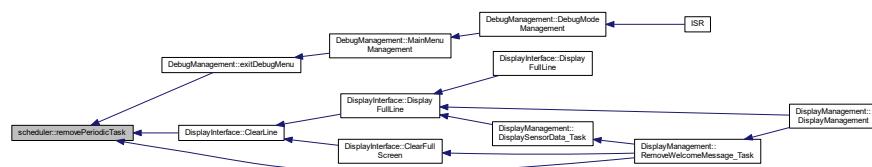
TRUE if the task has been removed, FALSE if the task does not exist in the scheduler

Definition at line 119 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.15.4.7 startScheduling()

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

Starts the tasks scheduling.

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

Returns

Nothing

Definition at line 93 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.15.4.8 updateTaskPeriod()

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

Task period update function.

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

Parameters

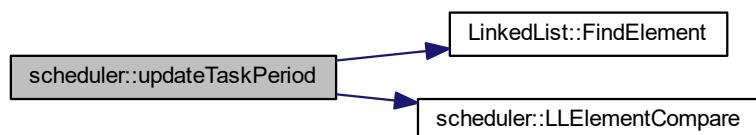
in	<i>task_ptr</i>	Pointer of the task to update
in	<i>period</i>	New period of the task

Returns

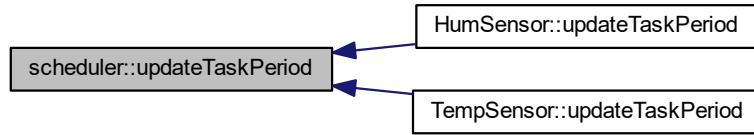
True if the update has been correctly done, false otherwise

Definition at line 142 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.15.5 Member Data Documentation

4.15.5.1 pit_number

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

Counter of periodic interrupts

Definition at line 140 of file scheduler.h.

4.15.5.2 task_count

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

Number of task in scheduler

Definition at line 136 of file scheduler.h.

4.15.5.3 TasksLL_ptr

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

Pointer to the linked list object containing the tasks

Definition at line 138 of file scheduler.h.

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

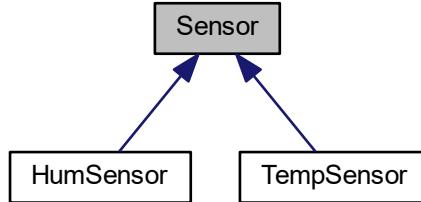
- [scheduler.h](#)
- [scheduler.cpp](#)

4.16 Sensor Class Reference

Generic class for sensor device.

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

Inheritance diagram for Sensor:



Public Member Functions

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

Static Public Member Functions

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

Protected Attributes

- bool `validity`
- bool `validity_last_read`
- uint32_t `valid_pit`
- uint16_t `validity_tmo`
- uint16_t `raw_data`
- uint16_t `valid_value`
- uint16_t `task_period`

4.16.1 Detailed Description

Generic class for sensor device.

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

Definition at line 18 of file Sensor.h.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 `Sensor()` [1/2]

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

`Sensor` class constructor.

This function initializes the class.

Returns

Nothing

Definition at line 22 of file Sensor.cpp.

4.16.2.2 `Sensor()` [2/2]

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

Overloaded sensor class constructor.

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

Parameters

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

Returns

Nothing

Definition at line 38 of file Sensor.cpp.

4.16.3 Member Function Documentation

4.16.3.1 getRawDataPtr()

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

Get pointer to raw sensor data.

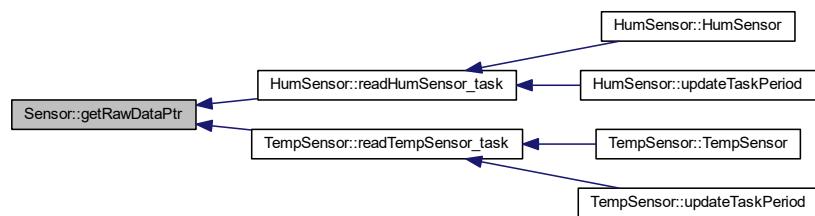
This function returns a pointer to the class member `raw_data`

Returns

Pointer to `raw_data`

Definition at line 53 of file Sensor.h.

Here is the caller graph for this function:



4.16.3.2 getTaskPeriod()

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

Task period get function.

This function returns the period of the sensor task

Returns

Period of the task (ms)

Definition at line 137 of file Sensor.h.

4.16.3.3 getValidity()

```
bool Sensor::getValidity ( ) [inline]
```

Data validity get function.

This function returns the validity of the sensor data

Returns

True if the sensor values are valid, false otherwise

Definition at line 117 of file Sensor.h.

4.16.3.4 getValue()

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

Get sensor value function.

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

Parameters

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

Returns

Validity

Definition at line 64 of file Sensor.h.

4.16.3.5 `getValueDecimal()`

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

Data formatting function - Decimal part.

This function return the decimal part of the sensor value

Returns

Decimal value of the sensor data

Definition at line 106 of file Sensor.h.

4.16.3.6 `getValueInteger()`

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

Data formatting function - Integer part.

This function return the integer part of the sensor value

Returns

Integer value of the sensor data

Definition at line 95 of file Sensor.h.

4.16.3.7 `readSensor_task()`

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

Task for reading sensor values.

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

Returns

Nothing

Definition at line 46 of file Sensor.h.

4.16.3.8 `setLastValidity()`

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

Validity setting function.

This function sets the class member `validity_last_read`

Parameters

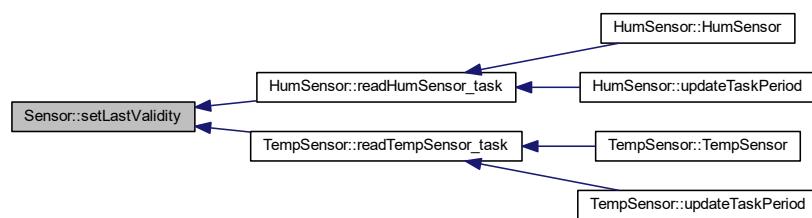
in	<i>validity</i>	Value of validity
----	-----------------	-------------------

Returns

Nothing

Definition at line 76 of file Sensor.h.

Here is the caller graph for this function:

**4.16.3.9 setValidityTMO()**

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

Validity timeout setting function.

This function sets the validity timeout.

Parameters

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

Returns

Nothing

Definition at line 149 of file Sensor.h.

4.16.3.10 updateTaskPeriod()

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

Task period update.

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

Parameters

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

Returns

True if the period has been updated, false otherwise

Definition at line 129 of file Sensor.h.

4.16.3.11 updateValidData()

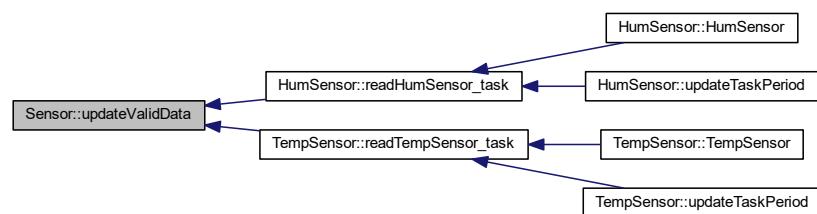
```
void Sensor::updateValidData ( )
```

Definition at line 53 of file Sensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.16.4 Member Data Documentation

4.16.4.1 raw_data

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

Raw value of sensor data (directly coming from driver)

Definition at line 161 of file Sensor.h.

4.16.4.2 task_period

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

Task period

Definition at line 165 of file Sensor.h.

4.16.4.3 valid坑

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

pit number of the last time when data were valid

Definition at line 158 of file Sensor.h.

4.16.4.4 valid_value

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

Valid value of sensor data

Definition at line 163 of file Sensor.h.

4.16.4.5 validity

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

Validity of sensor data

Definition at line 155 of file Sensor.h.

4.16.4.6 validity_last_read

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

Validity of last read sensor data

Definition at line 156 of file Sensor.h.

4.16.4.7 validity_tmo

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

Number of PITs after which the sensor value is declared invalid

Definition at line 159 of file Sensor.h.

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

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

4.17 SensorManagement Class Reference

[Sensor](#) management class.

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

Public Member Functions

- [SensorManagement \(\)](#)
Class constructor.
- [uint8_t getSensorCount \(\)](#)
Sensor count get function.
- [bool updateTaskPeriod \(uint16_t period\)](#)
Sensors tasks period update.
- [void getFullStringFormattedValue \(uint8_t sensor_idx, String *str\)](#)
Sensor value formatting function.
- [void * getSensorObjectPtr \(T_SensorManagement_Sensor_Type type\)](#)
Sensor object pointer get function.

Private Attributes

- [uint8_t nb_sensors](#)
- [void ** sensor_ptr_table](#)

4.17.1 Detailed Description

[Sensor](#) management class.

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

Definition at line 29 of file SensorManagement.h.

4.17.2 Constructor & Destructor Documentation

4.17.2.1 SensorManagement()

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

Class constructor.

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

Returns

Nothing

Definition at line 26 of file SensorManagement.cpp.

4.17.3 Member Function Documentation

4.17.3.1 getFullStringFormattedValue()

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

[Sensor](#) value formatting function.

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

Parameters

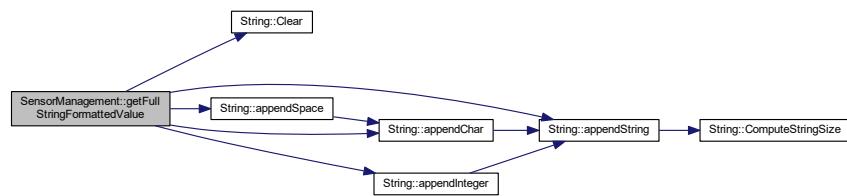
in	<i>sensor_idx</i>	Index of the requested sensor
out	<i>str</i>	Pointer to the formatted string

Returns

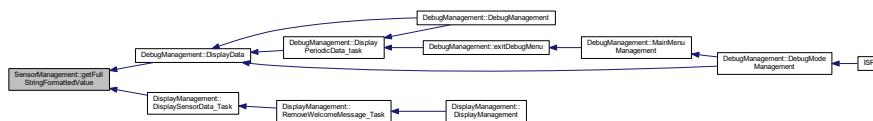
Nothing

Definition at line 70 of file SensorManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.17.3.2 getSensorCount()**

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

Sensor count get function.

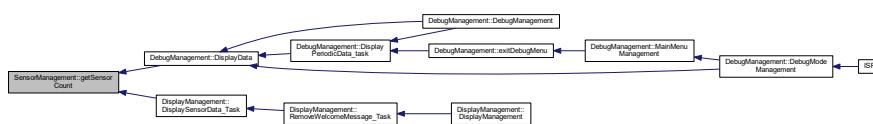
This function returns the number of sensors present in the SW

Returns

Number of sensors.

Definition at line 47 of file SensorManagement.h.

Here is the caller graph for this function:



4.17.3.3 getSensorObjectPtr()

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

Sensor object pointer get function.

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

Parameters

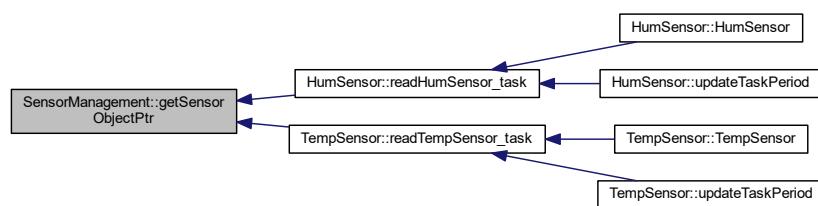
<code>in</code>	<code>type</code>	Type of sensor to find
-----------------	-------------------	------------------------

Returns

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

Definition at line 90 of file SensorManagement.cpp.

Here is the caller graph for this function:

**4.17.3.4 updateTaskPeriod()**

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

Sensors tasks period update.

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

Parameters

<code>in</code>	<code>period</code>	New period.
-----------------	---------------------	-------------

Returns

True if the period has been updated, false otherwise.

Definition at line 53 of file SensorManagement.cpp.

4.17.4 Member Data Documentation

4.17.4.1 nb_sensors

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

Number of sensors

Definition at line 82 of file SensorManagement.h.

4.17.4.2 sensor_ptr_table

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

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

Definition at line 83 of file SensorManagement.h.

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

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

4.18 String Class Reference

[String](#) management class.

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

Public Member Functions

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

Private Member Functions

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

Private Attributes

- `uint8_t * string`
- `uint8_t size`

4.18.1 Detailed Description

`String` management class.

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

Definition at line 18 of file String.h.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 `String()` [1/2]

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

Class constructor.

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

Parameters

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

Returns

Nothing

Definition at line 15 of file String.cpp.

Here is the call graph for this function:



4.18.2.2 String() [2/2]

`String::String ()`

Class constructor.

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

Returns

Nothing

Definition at line 33 of file String.cpp.

4.18.2.3 ~String()

`String::~String ()`

Class destructor.

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

Returns

Nothing

Definition at line 39 of file String.cpp.

Here is the call graph for this function:



4.18.3 Member Function Documentation

4.18.3.1 appendBool()

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

Boolean adding function.

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

Parameters

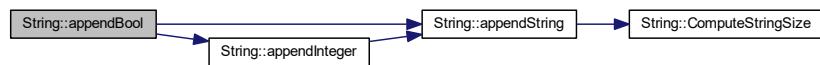
in	<code>data</code>	Boolean data to add
in	<code>isText</code>	Defines the conversion mode : text or integer

Returns

Nothing

Definition at line 121 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.18.3.2 appendChar()

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

Character adding function.

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

Parameters

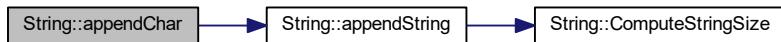
in	<i>data</i>	1-byte character to add
----	-------------	-------------------------

Returns

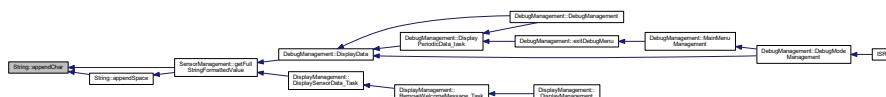
Nothing

Definition at line 139 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.18.3.3 appendInteger()

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

Integer adding function.

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

Parameters

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

Returns

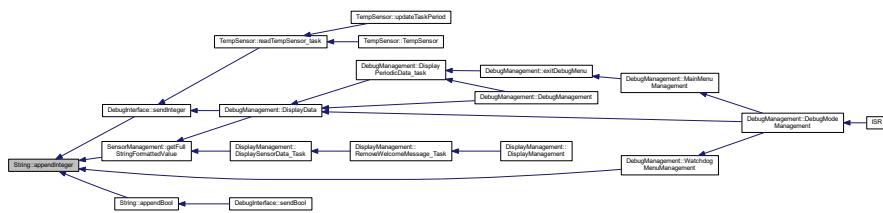
Nothing

Definition at line 95 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.18.3.4 appendSpace()

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

Space adding function.

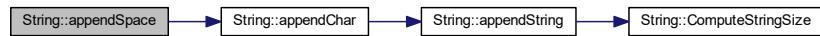
This function adds a space at the end of the string. It only calls appendChar function.

Returns

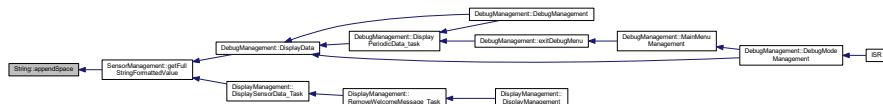
Nothing

Definition at line 123 of file String.h.

Here is the call graph for this function:



Here is the caller graph for this function:



4.18.3.5 appendString()

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

[String](#) adding function.

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

Parameters

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

Returns

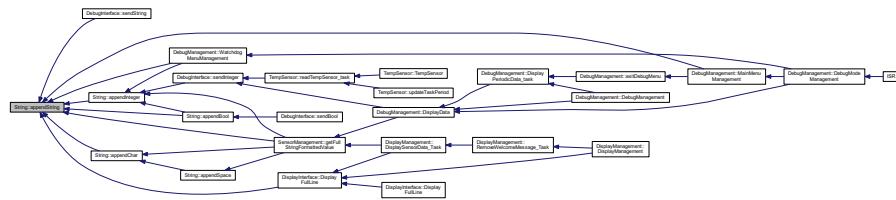
Nothing

Definition at line 57 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.18.3.6 Clear()**

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

String clear function.

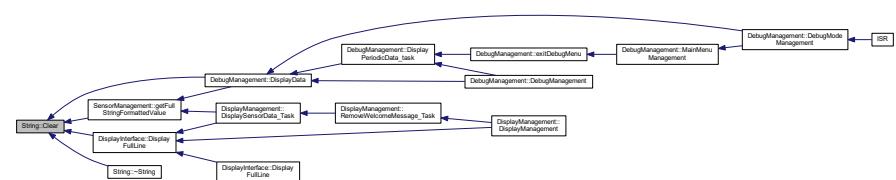
This function clears the string. Size is set to 0 and the memory is freed.

Returns

Nothing

Definition at line 113 of file String.cpp.

Here is the caller graph for this function:



4.18.3.7 ComputeStringSize()

```
uint8_t String::ComputeStringSize (
```

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

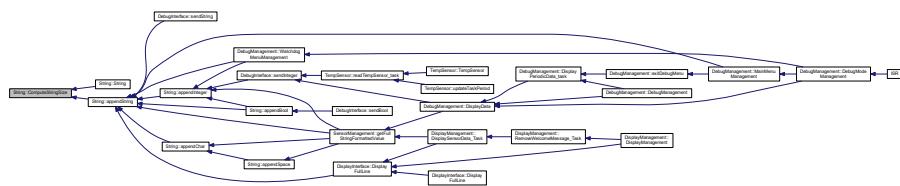
in *str* Pointer to the beginning of the string

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:



4.18.3.8 getSize()

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

Size get function.

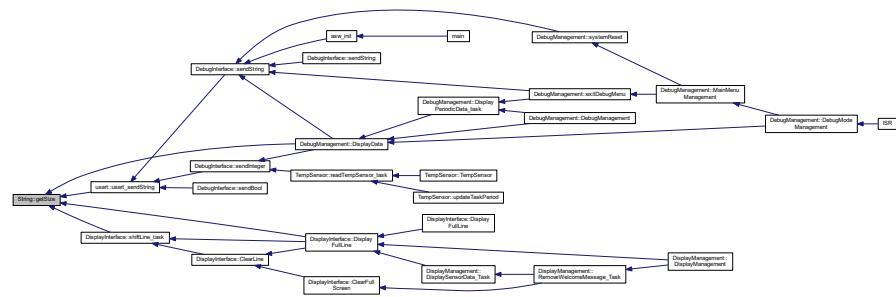
This function returns the size of the string.

Returns

Size of the string

Definition at line 64 of file String.h.

Here is the caller graph for this function:



4.18.3.9 `getString()`

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

String pointer get function.

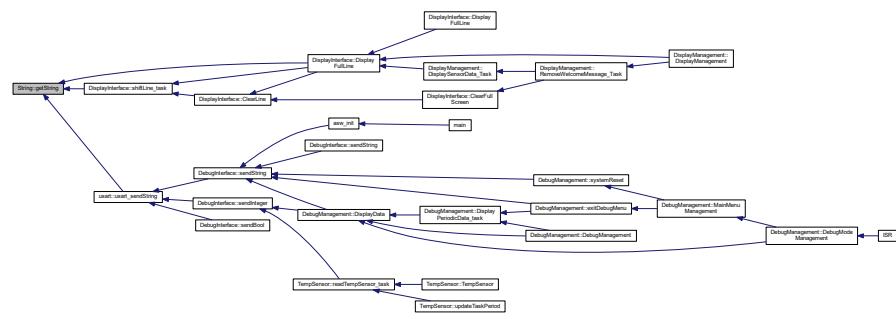
This function returns the pointer to the beginning of the string.

Returns

String pointer

Definition at line 53 of file String.h.

Here is the caller graph for this function:



4.18.4 Member Data Documentation

4.18.4.1 size

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

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

Definition at line 132 of file String.h.

4.18.4.2 string

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

Pointer to the start of the string

Definition at line 131 of file String.h.

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

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

4.19 T_ASW_init_cnf Struct Reference

ASW initialization configuration structure.

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

Public Attributes

- bool [isDebugEnabled](#)
- bool [isLEDActivated](#)
- bool [isSensorMgtActivated](#)
- bool [isDisplayActivated](#)

4.19.1 Detailed Description

ASW initialization configuration structure.

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

Definition at line 17 of file asw.h.

4.19.2 Member Data Documentation

4.19.2.1 isDebugActivated

bool T_ASW_init_cnf::isDebugActivated

Debug services activation flag

Definition at line 19 of file asw.h.

4.19.2.2 isDisplayActivated

bool T_ASW_init_cnf::isDisplayActivated

LCD display activation flag

Definition at line 22 of file asw.h.

4.19.2.3 isLEDActivated

bool T_ASW_init_cnf::isLEDActivated

Keep-alive LED activation flag

Definition at line 20 of file asw.h.

4.19.2.4 isSensorMgtActivated

bool T_ASW_init_cnf::isSensorMgtActivated

Sensor activation

Definition at line 21 of file asw.h.

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

- [asw.h](#)

4.20 Bmp180::T_BMP180_calib_data Struct Reference

Structure defining the calibration data of BMP180 sensor.

Public Attributes

- int16_t AC1
- int16_t AC2
- int16_t AC3
- uint16_t AC4
- uint16_t AC5
- uint16_t AC6
- int16_t B1
- int16_t B2
- int16_t MB
- int16_t MC
- int16_t MD

4.20.1 Detailed Description

Structure defining the calibration data of BMP180 sensor.

Definition at line 147 of file Bmp180.h.

4.20.2 Member Data Documentation

4.20.2.1 AC1

```
int16_t Bmp180::T_BMP180_calib_data::AC1
```

Definition at line 149 of file Bmp180.h.

4.20.2.2 AC2

```
int16_t Bmp180::T_BMP180_calib_data::AC2
```

Definition at line 150 of file Bmp180.h.

4.20.2.3 AC3

```
int16_t Bmp180::T_BMP180_calib_data::AC3
```

Definition at line 151 of file Bmp180.h.

4.20.2.4 AC4

```
uint16_t Bmp180::T_BMP180_calib_data::AC4
```

Definition at line 152 of file Bmp180.h.

4.20.2.5 AC5

```
uint16_t Bmp180::T_BMP180_calib_data::AC5
```

Definition at line 153 of file Bmp180.h.

4.20.2.6 AC6

```
uint16_t Bmp180::T_BMP180_calib_data::AC6
```

Definition at line 154 of file Bmp180.h.

4.20.2.7 B1

```
int16_t Bmp180::T_BMP180_calib_data::B1
```

Definition at line 155 of file Bmp180.h.

4.20.2.8 B2

```
int16_t Bmp180::T_BMP180_calib_data::B2
```

Definition at line 156 of file Bmp180.h.

4.20.2.9 MB

```
int16_t Bmp180::T_BMP180_calib_data::MB
```

Definition at line 157 of file Bmp180.h.

4.20.2.10 MC

```
int16_t Bmp180::T_BMP180_calib_data::MC
```

Definition at line 158 of file Bmp180.h.

4.20.2.11 MD

```
int16_t Bmp180::T_BMP180_calib_data::MD
```

Definition at line 159 of file Bmp180.h.

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

- [Bmp180.h](#)

4.21 Bmp180::T_BMP180_measurement_data Struct Reference

Structure defining a sensor value and its status.

Public Attributes

- `uint16_t value`
- `bool ready`
- `uint32_t ts`

4.21.1 Detailed Description

Structure defining a sensor value and its status.

Definition at line 168 of file Bmp180.h.

4.21.2 Member Data Documentation

4.21.2.1 ready

```
bool Bmp180::T_BMP180_measurement_data::ready
```

Definition at line 171 of file Bmp180.h.

4.21.2.2 ts

```
uint32_t Bmp180::T_BMP180_measurement_data::ts
```

Definition at line 172 of file Bmp180.h.

4.21.2.3 value

```
uint16_t Bmp180::T_BMP180_measurement_data::value
```

Definition at line 170 of file Bmp180.h.

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

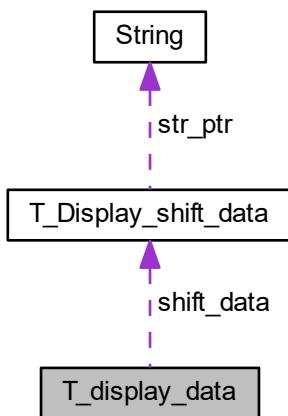
- [Bmp180.h](#)

4.22 T_display_data Struct Reference

Structure containing display data.

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

Collaboration diagram for T_display_data:



Public Attributes

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

4.22.1 Detailed Description

Structure containing display data.

This structure contains all data used for screen display

Definition at line 57 of file DisplayInterface.h.

4.22.2 Member Data Documentation

4.22.2.1 alignment

```
T_DisplayInterface_LineAlignment T_display_data::alignment
```

Line alignment

Definition at line 61 of file DisplayInterface.h.

4.22.2.2 display_str

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

Current string displayed on the screen

Definition at line 63 of file DisplayInterface.h.

4.22.2.3 isEmpty

```
bool T_display_data::isEmpty
```

Flag indicating if the line is empty or not

Definition at line 59 of file DisplayInterface.h.

4.22.2.4 mode

```
T_DisplayInterface_LineDisplayMode T_display_data::mode
```

Current display mode

Definition at line 60 of file DisplayInterface.h.

4.22.2.5 shift_data

```
T_Display_shift_data T_display_data::shift_data
```

Shift data for the current line

Definition at line 62 of file DisplayInterface.h.

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

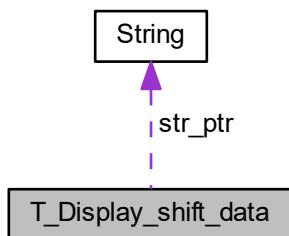
- [DisplayInterface.h](#)

4.23 T_Display_shift_data Struct Reference

Structure containing shift data.

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

Collaboration diagram for T_Display_shift_data:



Public Attributes

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

4.23.1 Detailed Description

Structure containing shift data.

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

Definition at line 45 of file DisplayInterface.h.

4.23.2 Member Data Documentation

4.23.2.1 str_cur_ptr

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

Pointer to the address of the first displayed character

Definition at line 48 of file DisplayInterface.h.

4.23.2.2 str_ptr

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

Pointer to the start address of the string

Definition at line 47 of file DisplayInterface.h.

4.23.2.3 temporization

```
uint8_t T_Display_shift_data::temporization
```

Shifting period

Definition at line 49 of file DisplayInterface.h.

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

- [DisplayInterface.h](#)

4.24 T_LCD_conf_struct Struct Reference

Structure defining [LCD](#) configuration.

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

Public Attributes

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

4.24.1 Detailed Description

Structure defining [LCD](#) configuration.

Definition at line 128 of file LCD.h.

4.24.2 Member Data Documentation

4.24.2.1 `backlight_en`

```
bool T_LCD_conf_struct::backlight_en
```

Screen backlight enable flag

Definition at line 132 of file LCD.h.

4.24.2.2 `cursor_en`

```
bool T_LCD_conf_struct::cursor_en
```

Screen cursor enable flag

Definition at line 136 of file LCD.h.

4.24.2.3 `cursorBlink_en`

```
bool T_LCD_conf_struct::cursorBlink_en
```

Screen cursor blinking enable flag

Definition at line 137 of file LCD.h.

4.24.2.4 `display_en`

`bool T_LCD_conf_struct::display_en`

Screen display enable flag

Definition at line 135 of file LCD.h.

4.24.2.5 `entryModeDir`

`bool T_LCD_conf_struct::entryModeDir`

Entry mode direction configuration

Definition at line 138 of file LCD.h.

4.24.2.6 `entryModeShift`

`bool T_LCD_conf_struct::entryModeShift`

Entry mode shift configuration

Definition at line 139 of file LCD.h.

4.24.2.7 `fontType_cnf`

`bool T_LCD_conf_struct::fontType_cnf`

Font configuration

Definition at line 134 of file LCD.h.

4.24.2.8 `i2c_addr`

`uint8_t T_LCD_conf_struct::i2c_addr`

I²C address if the screen

Definition at line 131 of file LCD.h.

4.24.2.9 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.

4.24.2.10 lineNumber_cnf

`bool T_LCD_conf_struct::lineNumber_cnf`

Screen line number configuration (1 or 2 lines)

Definition at line 133 of file LCD.h.

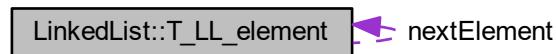
The documentation for this struct was generated from the following file:

- [LCD.h](#)

4.25 LinkedList::T_LL_element Struct Reference

Type defining a linked list element.

Collaboration diagram for LinkedList::T_LL_element:



Public Attributes

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

4.25.1 Detailed Description

Type defining a linked list element.

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

Definition at line 117 of file LinkedList.h.

4.25.2 Member Data Documentation

4.25.2.1 data_ptr

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

Definition at line 119 of file LinkedList.h.

4.25.2.2 nextElement

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

Definition at line 120 of file LinkedList.h.

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

- [LinkedList.h](#)

4.26 T_SensorManagement_Sensor_Config Struct Reference

[Sensor](#) informations structure.

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

Public Attributes

- [T_SensorManagement_Sensor_Type](#) `sensor_type`
- `uint16_t` `period`
- `uint16_t` `validity_tmo`
- `uint8_t *` `data_name_str`
- `uint8_t *` `unit_str`

4.26.1 Detailed Description

[Sensor](#) informations structure.

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

Definition at line 17 of file sensor_configuration.h.

4.26.2 Member Data Documentation

4.26.2.1 data_name_str

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

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

4.26.2.2 period

```
uint16_t T_SensorManagement_Sensor_Config::period
```

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

4.26.2.3 sensor_type

```
T_SensorManagement_Sensor_Type T_SensorManagement_Sensor_Config::sensor_type
```

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

4.26.2.4 unit_str

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

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

4.26.2.5 validity_tmo

```
uint16_t T_SensorManagement_Sensor_Config::validity_tmo
```

Definition at line 21 of file `sensor_configuration.h`.

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

- [sensor_configuration.h](#)

4.27 scheduler::Task_t Struct Reference

Type defining a task structure.

Public Attributes

- `TaskPtr_t TaskPtr`
- `uint16_t period`

4.27.1 Detailed Description

Type defining a task structure.

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

Definition at line 129 of file scheduler.h.

4.27.2 Member Data Documentation

4.27.2.1 period

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

Period of the task

Definition at line 132 of file scheduler.h.

4.27.2.2 TaskPtr

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

Pointer to the task

Definition at line 131 of file scheduler.h.

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

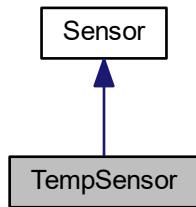
- `scheduler.h`

4.28 TempSensor Class Reference

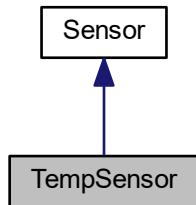
Class for temperature sensor.

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

Inheritance diagram for TempSensor:



Collaboration diagram for TempSensor:



Public Member Functions

- [TempSensor \(\)](#)
Class constructor.
- [TempSensor \(uint16_t val_tmo, uint16_t period\)](#)
Overloaded class constructor.
- [bool updateTaskPeriod \(uint16_t period\)](#)
Task period update.

Static Public Member Functions

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

Additional Inherited Members

4.28.1 Detailed Description

Class for temperature sensor.

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

Definition at line 20 of file TempSensor.h.

4.28.2 Constructor & Destructor Documentation

4.28.2.1 TempSensor() [1/2]

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

Class constructor.

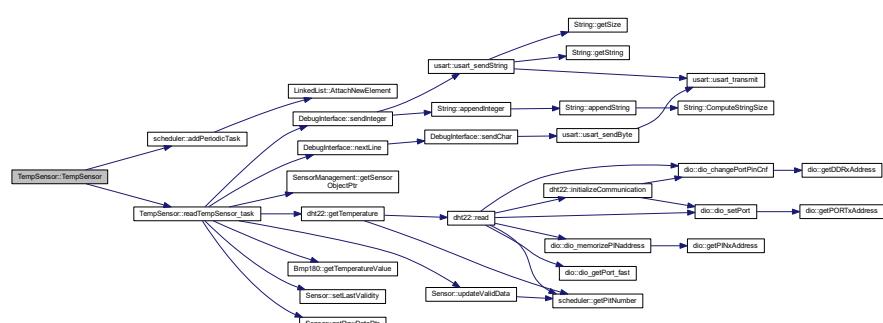
This function initializes all data of the class `TempSensor`. If needed, it creates a new instance of the DHT22 and BMP180 sensors objects. It also adds the periodic task in the scheduler.

Returns

Nothing

Definition at line 31 of file TempSensor.cpp.

Here is the call graph for this function:



4.28.2.2 TempSensor() [2/2]

```
TempSensor::TempSensor (
```

uint16_t val_tmo,

uint16_t period)

Overloaded class constructor.

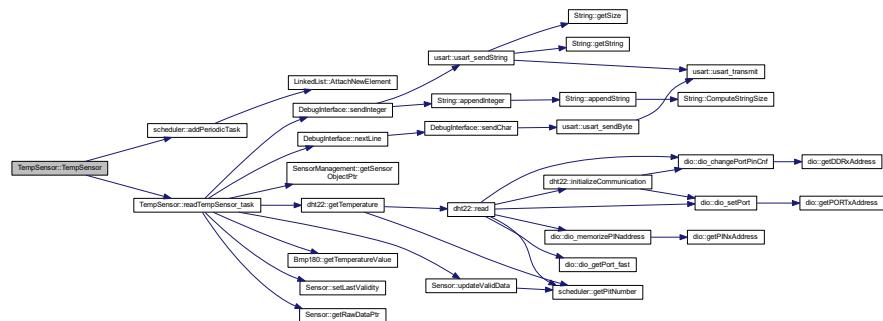
This function initializes all data of the class `TempSensor`. It sets validity timeout and task period to the given value. If needed, it creates a new instance of the DHT22 and BMP180 sensor objects. It also adds the periodic task in the scheduler.

Returns

Nothing

Definition at line 46 of file TempSensor.cpp.

Here is the call graph for this function:



4.28.3 Member Function Documentation

4.28.3.1 readTempSensor_task()

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

Task for reading temperature values.

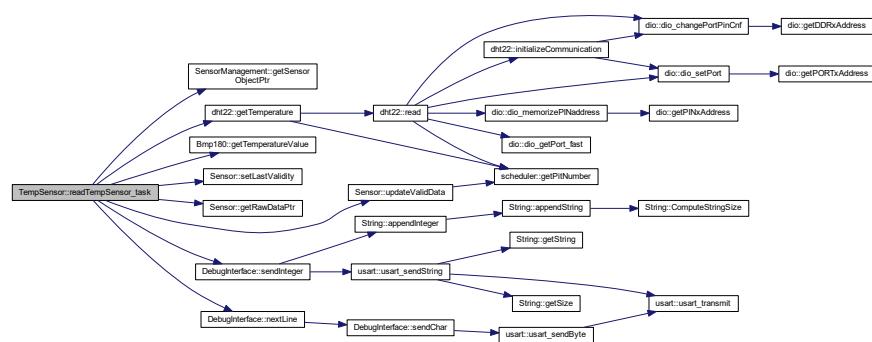
This task reads temperature data using DHT22 and BMP180 drivers. It is called periodically. The returned temperature is the mean between both sensors values, if only one sensor is valid, only this value is used .

Returns

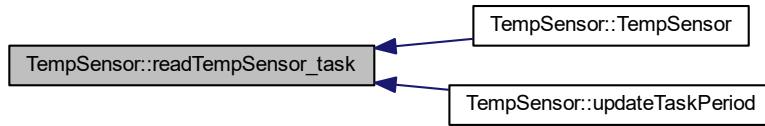
Nothing

Definition at line 60 of file TempSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.28.3.2 updateTaskPeriod()

```
bool TempSensor::updateTaskPeriod (
```

Task period update.

This function updates the period of the temperature task.

Parameters

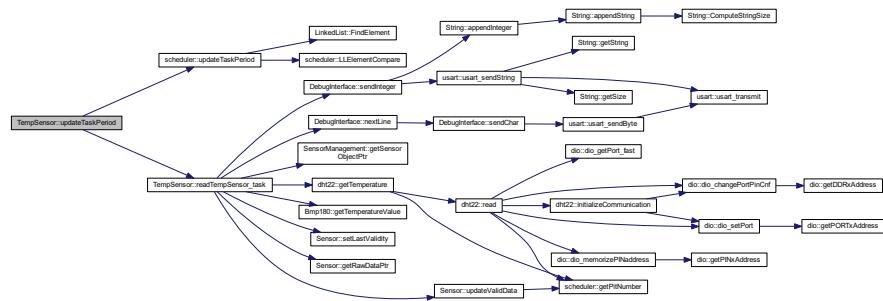
in *period* New period of the task

Returns

True if the period has been updated, false otherwise

Definition at line 117 of file TempSensor.cpp.

Here is the call graph for this function:



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

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

4.29 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 configureTimer3 \(uint16_t a_prescaler, uint16_t a_ctcValue\)](#)
Configures Timer #3.
- [void startTimer1 \(\)](#)
Start Timer #1.
- [void startTimer3 \(\)](#)
Start Timer #3.
- [void stopTimer1 \(\)](#)
Stops Timer #1.
- [void stopTimer3 \(\)](#)
Stops Timer #3.
- [uint16_t getTimer1Value \(\)](#)
Reads current value of timer #1.

Private Attributes

- `uint8_t prescaler1`
- `uint8_t prescaler3`

4.29.1 Detailed Description

Class defining a timer.

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

Definition at line 19 of file timer.h.

4.29.2 Constructor & Destructor Documentation

4.29.2.1 timer()

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

Class constructor.

This function initializes class attributes

Returns

Nothing

Definition at line 15 of file timer.cpp.

4.29.3 Member Function Documentation

4.29.3.1 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 21 of file timer.cpp.

Here is the caller graph for this function:

**4.29.3.2 configureTimer3()**

```
void timer::configureTimer3 (
    uint16_t a_prescaler,
    uint16_t a_ctcValue )
```

Configures Timer #3.

This function configures hardware timer #3 in CTC mode, enables its interrupts, sets prescaler to *a_prescaler* and CTC value to *a_ctcValue*

Parameters

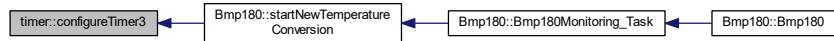
in	<i>a_prescaler</i>	prescaler value
in	<i>a_ctcValue</i>	Value to which the counter will compare before raising an interrupt

Returns

Nothing

Definition at line 58 of file timer.cpp.

Here is the caller graph for this function:



4.29.3.3 getTimer1Value()

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

Reads current value of timer #1.

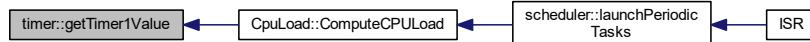
This function reads the value of of timer #1 using register TCNT1. The function is inlined to speed up SW execution.

Returns

Current timer value

Definition at line 81 of file timer.h.

Here is the caller graph for this function:



4.29.3.4 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 96 of file timer.cpp.

Here is the caller graph for this function:



4.29.3.5 startTimer3()

```
void timer::startTimer3 ( )
```

Start Timer #3.

This functions starts Timer #3. Timer shall be initialized before this function is called.

Returns

Nothing

Definition at line 107 of file timer.cpp.

Here is the caller graph for this function:



4.29.3.6 stopTimer1()

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

Stops Timer #1.

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

Returns

Nothing

Definition at line 118 of file timer.cpp.

4.29.3.7 stopTimer3()

```
void timer::stopTimer3 ( )
```

Stops Timer #3.

This functions stops timer #3 by resetting bits 0-2 of TCCR3B

Returns

Nothing

Definition at line 125 of file timer.cpp.

Here is the caller graph for this function:



4.29.4 Member Data Documentation

4.29.4.1 prescaler1

```
uint8_t timer::prescaler1 [private]
```

Definition at line 87 of file timer.h.

4.29.4.2 prescaler3

```
uint8_t timer::prescaler3 [private]
```

Definition at line 88 of file timer.h.

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

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

4.30 usart Class Reference

USART serial bus class.

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

Public Member Functions

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

Private Member Functions

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

Private Attributes

- uint16_t [BaudRate](#)

4.30.1 Detailed Description

USART serial bus class.

This class defines all useful functions for USART serial bus

Definition at line 16 of file usart.h.

4.30.2 Constructor & Destructor Documentation

4.30.2.1 usart()

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

Class usart constructor.

Initializes the class and call hardware initialization function

Parameters

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

Returns

Nothing.

Definition at line 18 of file usart.cpp.

Here is the call graph for this function:



4.30.3 Member Function Documentation

4.30.3.1 setBaudRate()

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

Setting baud rate.

This function sets the attribute BaudRate of the class usart

Parameters

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

Returns

Nothing

Definition at line 74 of file usart.cpp.

4.30.3.2 usart_init()

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

USART hardware initialization.

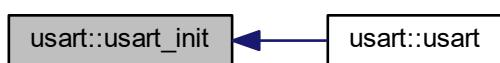
This function will initialize the USART using selected baudrate. User must pay attention to select one of the usually used Baud Rate (9600, 19200, 38400, 57600). Note that since an uint16 is used as argument, Baud rate cannot be more than 57600.

Returns

Nothing.

Definition at line 25 of file usart.cpp.

Here is the caller graph for this function:



4.30.3.3 usart_read()

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

USART read function.

This function will read reception register of USART

Returns

The function returns the 8 bits read from reception buffer

Definition at line 90 of file usart.cpp.

4.30.3.4 usart_sendByte()

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

Send a single byte on USART link.

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

Parameters

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

Returns

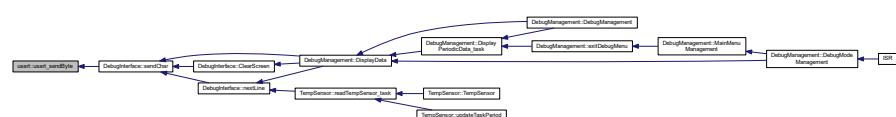
Nothing.

Definition at line 68 of file usart.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.30.3.5 usart_sendString()

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

Send a string on USART link.

This function writes the string object data to the serial link using usart_transmit function

Parameters

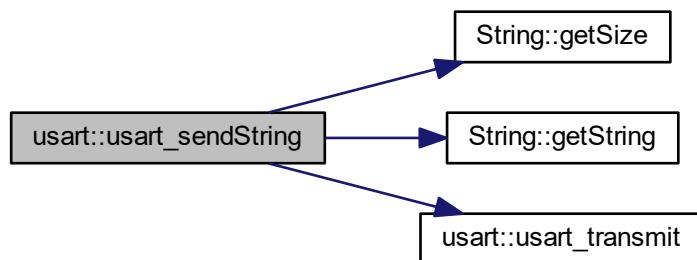
in	<i>str</i>	Pointer to the string being sent
----	------------	----------------------------------

Returns

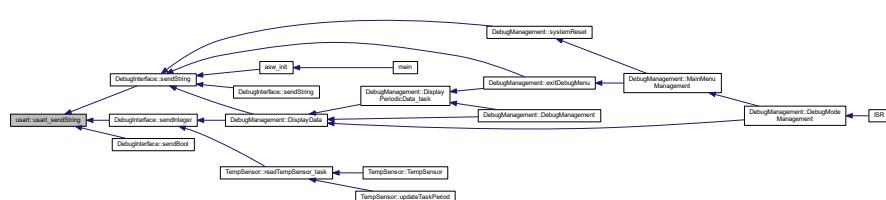
Nothing.

Definition at line 48 of file usart.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.30.3.6 usart_transmit()

```
void usart::usart_transmit (
```

USART Transmit data.

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

Parameters

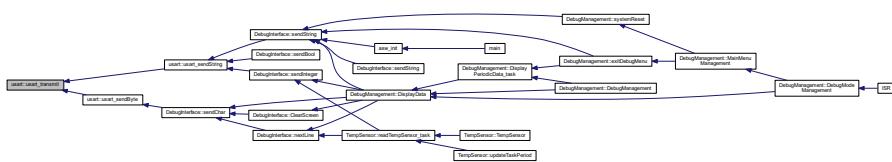
in	<i>Data</i>	Desired data char to transmit
----	-------------	-------------------------------

Returns

Nothing.

Definition at line 81 of file usart.cpp.

Here is the caller graph for this function:



4.30.4 Member Data Documentation

4.30.4.1 BaudRate

uint16_t usart::BaudRate [private]

Defines the baud rate used by driver

Definition at line 77 of file usart.h.

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

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

4.31 Watchdog Class Reference

Watchdog management class.

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

Public Member Functions

- [Watchdog \(\)](#)
Class constructor.
- [Watchdog \(uint8_t timeout\)](#)
Overloaded class constructor.
- [void reset \(\)](#)
Watchdog reset function.
- [void timeoutUpdate \(uint8_t value\)](#)
Watchdog timeout value update function.
- [void SystemReset \(\)](#)
System reset function.
- [uint16_t getTMOValue \(\)](#)
Watchdog timeout get value.
- [bool isEnabled \(\)](#)
Watchdog status function.
- [bool SwitchWdg \(\)](#)
Watchdog switching function.

Private Member Functions

- [void disable \(\)](#)
Watchdog disabling function.
- [void enable \(uint8_t value\)](#)
Watchdog enabling function.

Private Attributes

- [uint8_t tmo_value](#)
- [bool isActive](#)

4.31.1 Detailed Description

[Watchdog](#) management class.

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

Definition at line 31 of file Watchdog.h.

4.31.2 Constructor & Destructor Documentation

4.31.2.1 Watchdog() [1/2]

```
Watchdog::Watchdog ( )
```

Class constructor.

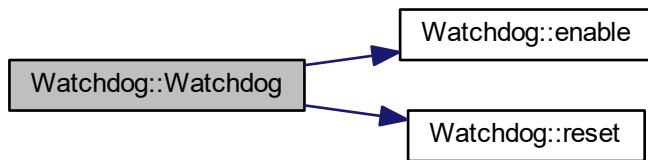
This function initializes the watchdog class. It enables the HW watchdog with a default timeout value.

Returns

Nothing

Definition at line 19 of file Watchdog.cpp.

Here is the call graph for this function:



4.31.2.2 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

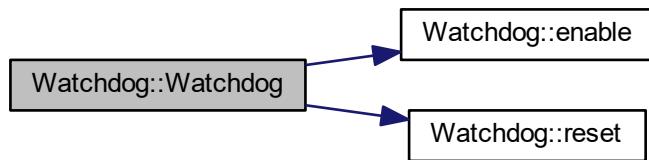
in	timeout	Timeout value requested for the watchdog
----	---------	--

Returns

Nothing

Definition at line 26 of file Watchdog.cpp.

Here is the call graph for this function:



4.31.3 Member Function Documentation

4.31.3.1 disable()

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

[Watchdog](#) disabling function.

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

Returns

Nothing

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

Here is the caller graph for this function:



4.31.3.2 enable()

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

[Watchdog](#) enabling function.

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

Parameters

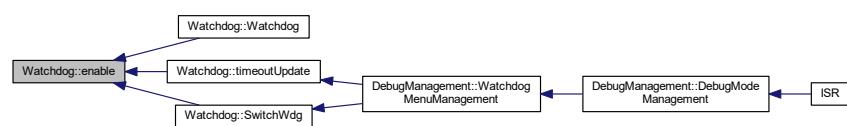
in	<i>value</i>	Timeout value
----	--------------	---------------

Returns

Nothing

Definition at line 34 of file Watchdog.cpp.

Here is the caller graph for this function:

**4.31.3.3 getTMOValue()**

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

[Watchdog](#) timeout get value.

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

Returns

Timeout value.

Definition at line 75 of file Watchdog.cpp.

Here is the caller graph for this function:



4.31.3.4 isEnabled()

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

[Watchdog](#) status function.

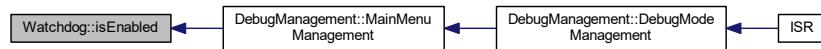
This function returns the current status of the watchdog : enabled or disabled.

Returns

True if the watchdog is enabled, false otherwise.

Definition at line 91 of file Watchdog.h.

Here is the caller graph for this function:



4.31.3.5 reset()

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

[Watchdog](#) reset function.

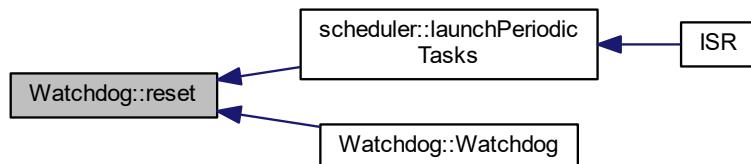
This function resets the watchdog timer by calling wdt_reset macro

Returns

Nothing

Definition at line 53 of file Watchdog.cpp.

Here is the caller graph for this function:



4.31.3.6 SwitchWdg()

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

[Watchdog](#) switching function.

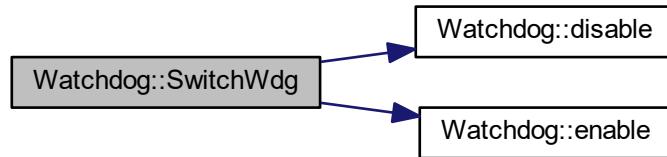
This function switches the state of the watchdog. If it was enabled, the function disables the watchdog, and if it was disabled, the function enables it with the memorized timeout value. The function returns the new status of the watchdog.

Returns

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

Definition at line 117 of file Watchdog.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.31.3.7 SystemReset()

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

System reset function.

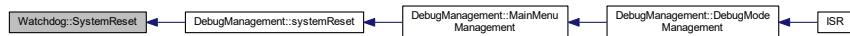
This function provokes a system reset by going in an infinite loop. Thus the watchdog will reset the CPU when the timeout occurs.

Returns

Nothing

Definition at line 70 of file Watchdog.cpp.

Here is the caller graph for this function:

**4.31.3.8 timeoutUpdate()**

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

[Watchdog](#) timeout value update function.

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

Parameters

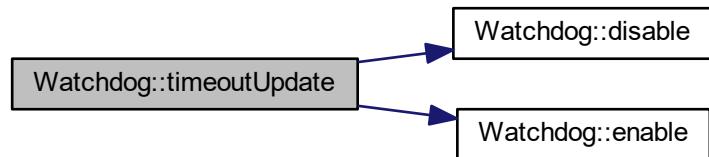
in	value	New timeout value
----	-------	-------------------

Returns

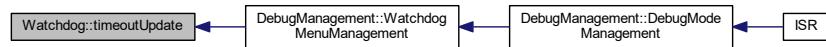
Nothing

Definition at line 58 of file Watchdog.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.31.4 Member Data Documentation

4.31.4.1 isActive

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

[Watchdog](#) activation flag

Definition at line 109 of file Watchdog.h.

4.31.4.2 tmo_value

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

Current timeout value

Definition at line 108 of file Watchdog.h.

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

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

Chapter 5

File Documentation

5.1 asw.cpp File Reference

ASW main file.

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

Include dependency graph for asw.cpp:

Include dependency graph for asw.cpp:



Functions

- void asw_init ()

Initialization of ASW.

5.1.1 Detailed Description

ASW main file.

Date

15 mars 2018

Author

nicls67

5.1.2 Function Documentation

5.1.2.1 asw_init()

```
void asw_init ( )
```

Initialization of ASW.

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

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

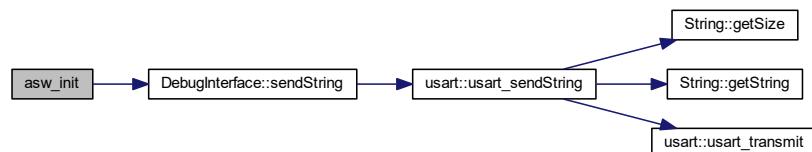
This function shall be called after BSW initialization function.

Returns

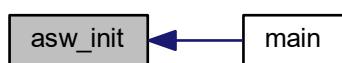
Nothing

Definition at line 35 of file asw.cpp.

Here is the call graph for this function:



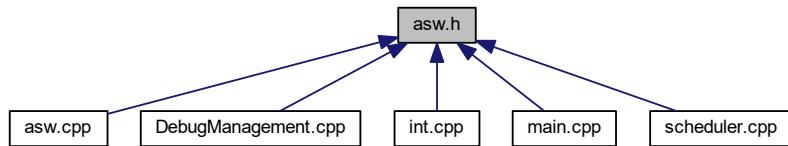
Here is the caller graph for this function:



5.2 asw.h File Reference

ASW main header file.

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



Classes

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

Functions

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

5.2.1 Detailed Description

ASW main header file.

Date

15 mars 2018

Author

nicls67

5.2.2 Function Documentation

5.2.2.1 asw_init()

```
void asw_init ( )
```

Initialization of ASW.

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

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

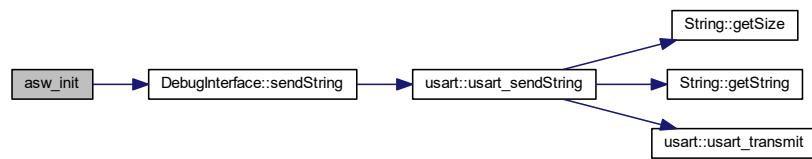
This function shall be called after BSW initialization function.

Returns

Nothing

Definition at line 35 of file asw.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



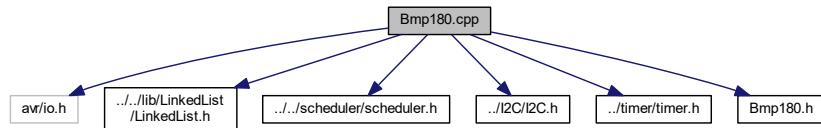
5.3 Bmp180.cpp File Reference

[Bmp180](#) class source file.

```
#include <avr/io.h>
#include "../../lib/LinkedList/LinkedList.h"
#include "../../scheduler/scheduler.h"
#include "../I2C/I2C.h"
#include "../timer/timer.h"
```

```
#include "Bmp180.h"
```

Include dependency graph for Bmp180.cpp:



Variables

- [Bmp180 * p_global_BSW_bmp180](#)

5.3.1 Detailed Description

[Bmp180](#) class source file.

Date

27 juil. 2019

Author

nicls67

5.3.2 Variable Documentation

5.3.2.1 p_global_BSW_bmp180

[Bmp180*](#) p_global_BSW_bmp180

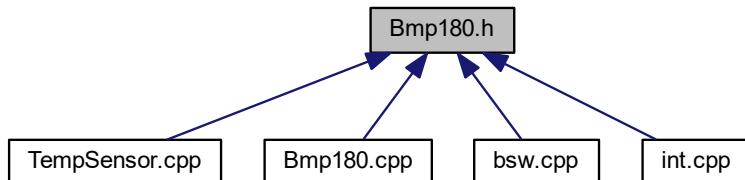
Pointer to BMP180 driver object

Definition at line 21 of file Bmp180.cpp.

5.4 Bmp180.h File Reference

[Bmp180](#) class header file.

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



Classes

- class [Bmp180](#)
BMP180 sensor class definition.
- struct [Bmp180::T_BMP180_calib_data](#)
Structure defining the calibration data of BMP180 sensor.
- struct [Bmp180::T_BMP180_measurement_data](#)
Structure defining a sensor value and its status.

Macros

- #define [BMP180_I2C_BITRATE](#) 100000
- #define [BMP180_I2C_ADDR](#) 0x77
- #define [BMP180_CHIP_ID_EEP_ADDR](#) 0xD0
- #define [BMP180_CHIP_ID_CALIB_EEP_START_ADDR](#) 0xAA
- #define [BMP180_CHIP_ID_EXPECTED](#) 0x55
- #define [BMP180_CTRL_MEAS_EEP_ADDR](#) 0xF4
- #define [BMP180_CTRL_MEAS_START_TEMP_CONV](#) 0x2E
- #define [BMP180_TIMER_PRESCALER_VALUE](#) 64
- #define [BMP180_TEMP_MEAS_TIMER_CTC_VALUE](#) ((F_CPU/PRESCALER_PERIODIC_TIMER)/(1000/10))
- #define [BMP180_OUT_REG_LSB EEPROM_ADDR](#) 0xF7
- #define [BMP180_OUT_REG_MSB EEPROM_ADDR](#) 0xF6
- #define [BMP180_MONITORING_DEFAULT_PERIOD](#) 500

Enumerations

- enum [T_BMP180_status](#) { [IDLE](#), [TEMP_CONV_IN_PROGRESS](#), [PRESSURE_CONV_IN_PROGRESS](#), [COMM_FAILED](#) }
- Enumeration defining the possible statuses for BMP180 sensor driver.*

Variables

- [Bmp180 * p_global_BSW_bmp180](#)

5.4.1 Detailed Description

[Bmp180](#) class header file.

Date

27 juil. 2019

Author

nicls67

5.4.2 Macro Definition Documentation

5.4.2.1 BMP180_CHIP_ID_CALIB_EEP_START_ADDR

```
#define BMP180_CHIP_ID_CALIB_EEP_START_ADDR 0xAA
```

EEPROM calibration data start address

Definition at line 17 of file Bmp180.h.

5.4.2.2 BMP180_CHIP_ID_EEP_ADDR

```
#define BMP180_CHIP_ID_EEP_ADDR 0xD0
```

Chip ID EEPROM address

Definition at line 16 of file Bmp180.h.

5.4.2.3 BMP180_CHIP_ID_EXPECTED

```
#define BMP180_CHIP_ID_EXPECTED 0x55
```

Expected chip ID

Definition at line 18 of file Bmp180.h.

5.4.2.4 BMP180_CTRL_MEAS_EEP_ADDR

```
#define BMP180_CTRL_MEAS_EEP_ADDR 0xF4
```

Address of the measurement control register in EEPROM

Definition at line 20 of file Bmp180.h.

5.4.2.5 BMP180_CTRL_MEAS_START_TEMP_CONV

```
#define BMP180_CTRL_MEAS_START_TEMP_CONV 0x2E
```

Value of measurement control register to start a temperature conversion

Definition at line 21 of file Bmp180.h.

5.4.2.6 BMP180_I2C_ADDR

```
#define BMP180_I2C_ADDR 0x77
```

I²C address of the sensor

Definition at line 14 of file Bmp180.h.

5.4.2.7 BMP180_I2C_BITRATE

```
#define BMP180_I2C_BITRATE 100000
```

Bitrate used for I²C communication

Definition at line 13 of file Bmp180.h.

5.4.2.8 BMP180_MONITORING_DEFAULT_PERIOD

```
#define BMP180_MONITORING_DEFAULT_PERIOD 500
```

Monitoring period is set by default to 500ms

Definition at line 29 of file Bmp180.h.

5.4.2.9 BMP180_OUT_REG_LSB_EEPROM_ADDR

```
#define BMP180_OUT_REG_LSB_EEPROM_ADDR 0xF7
```

Address of LSB out register

Definition at line 26 of file Bmp180.h.

5.4.2.10 BMP180_OUT_REG_MSB_EEPROM_ADDR

```
#define BMP180_OUT_REG_MSB_EEPROM_ADDR 0xF6
```

Address of MSB out register

Definition at line 27 of file Bmp180.h.

5.4.2.11 BMP180_TEMP_MEAS_TIMER_CTC_VALUE

```
#define BMP180_TEMP_MEAS_TIMER_CTC_VALUE ((F_CPU/PRESCALER_PERIODIC_TIMER)/(1000/10))
```

Compare value for periodic timer

Definition at line 24 of file Bmp180.h.

5.4.2.12 BMP180_TIMER_PRESCALER_VALUE

```
#define BMP180_TIMER_PRESCALER_VALUE 64
```

Value of prescaler to use for timer

Definition at line 23 of file Bmp180.h.

5.4.3 Enumeration Type Documentation**5.4.3.1 T_BMP180_status**

```
enum T_BMP180_status
```

Enumeration defining the possible statuses for BMP180 sensor driver.

Enumerator

IDLE	No conversion is in progress and communication is OK
TEMP_CONV_IN_PROGRESS	A temperature conversion is in progress
PRESSURE_CONV_IN_PROGRESS	A pressure conversion is in progress
COMM_FAILED	Communication is failed

Definition at line 34 of file Bmp180.h.

5.4.4 Variable Documentation

5.4.4.1 p_global_BSW_bmp180

Bmp180* p_global_BSW_bmp180

Pointer to BMP180 driver object

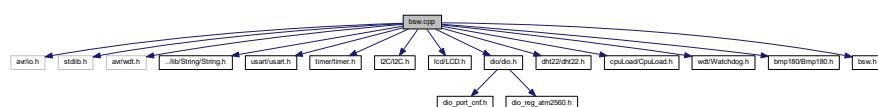
Definition at line 21 of file Bmp180.cpp.

5.5 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/uart.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 "bmp180/Bmp180.h"
#include "bsw.h"
```

Include dependency graph for bsw.cpp:



Functions

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

Initialization of BSW.

5.5.1 Detailed Description

BSW main file.

Date

13 mars 2018

Author

nicls67

5.5.2 Function Documentation

5.5.2.1 [bsw_init\(\)](#)

```
void bsw_init ( )
```

Initialization of BSW.

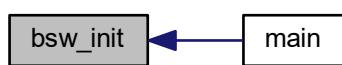
This function instantiates all driver objects, leading hardware initialization. The addresses of driver objects are then stored in BSW_cnf_struct structure.

Returns

Nothing

Definition at line 27 of file bsw.cpp.

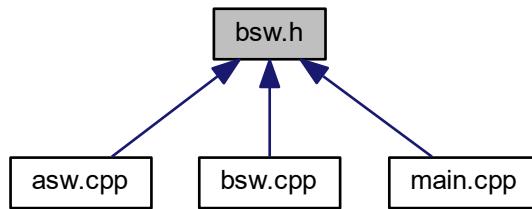
Here is the caller graph for this function:



5.6 bsw.h File Reference

BSW main header file.

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



Functions

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

Initialization of BSW.

5.6.1 Detailed Description

BSW main header file.

Date

13 mars 2018

Author

nicls67

5.6.2 Function Documentation

5.6.2.1 bsw_init()

```
void bsw_init ( )
```

Initialization of BSW.

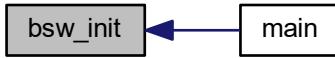
This function instantiates all driver objects, leading hardware initialization. The addresses of driver objects are then stored in BSW_cnf_struct structure.

Returns

Nothing

Definition at line 27 of file bsw.cpp.

Here is the caller graph for this function:

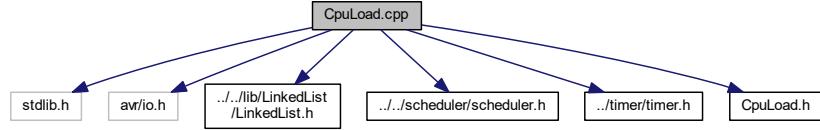


5.7 CpuLoad.cpp File Reference

Defines functions of class [CpuLoad](#).

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

Include dependency graph for CpuLoad.cpp:



Variables

- [CpuLoad * p_global_BSW_cpuload](#)

5.7.1 Detailed Description

Defines functions of class [CpuLoad](#).

Date

21 mars 2019

Author

nicls67

5.7.2 Variable Documentation

5.7.2.1 p_global_BSW_cpuload

`CpuLoad* p_global_BSW_cpuload`

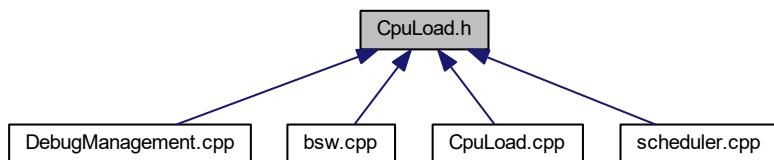
Pointer to cpu load library object

Definition at line 18 of file CpuLoad.cpp.

5.8 CpuLoad.h File Reference

[CpuLoad](#) class header file.

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



Classes

- class [CpuLoad](#)

Class defining CPU load libraries.

Macros

- `#define NB_OF_SAMPLES 50`

Variables

- `CpuLoad * p_global_BSW_cpuload`

5.8.1 Detailed Description

`CpuLoad` class header file.

Date

21 mars 2019

Author

nicls67

5.8.2 Macro Definition Documentation

5.8.2.1 NB_OF_SAMPLES

```
#define NB_OF_SAMPLES 50
```

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

5.8.3 Variable Documentation

5.8.3.1 p_global_BSW_cpuload

`CpuLoad* p_global_BSW_cpuload`

Pointer to cpu load library object

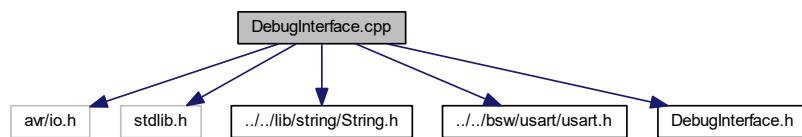
Definition at line 18 of file `CpuLoad.cpp`.

5.9 DebugInterface.cpp File Reference

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

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

Include dependency graph for DebugInterface.cpp:



Variables

- `DebugInterface * p_global_ASW_DebugInterface`

5.9.1 Detailed Description

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

Date

15 mars 2018

Author

nicls67

5.9.2 Variable Documentation

5.9.2.1 `p_global_ASW_DebugInterface`

`DebugInterface* p_global_ASW_DebugInterface`

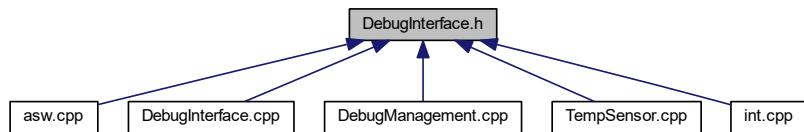
Pointer to USART debug interface object

Definition at line 19 of file DebugInterface.cpp.

5.10 DebugInterface.h File Reference

Header file for debug and logging functions.

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



Classes

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

Macros

- `#define USART_BAUDRATE (uint16_t)9600`

Variables

- `DebugInterface * p_global_ASW_DebugInterface`

5.10.1 Detailed Description

Header file for debug and logging functions.

Date

15 mars 2018

Author

nicls67

5.10.2 Macro Definition Documentation

5.10.2.1 USART_BAUDRATE

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

uart connection to PC uses a baud rate of 9600

Definition at line 15 of file DebugInterface.h.

5.10.3 Variable Documentation

5.10.3.1 p_global_ASW_DebugInterface

```
DebugInterface* p_global_ASW_DebugInterface
```

Pointer to USART debug interface object

Definition at line 19 of file DebugInterface.cpp.

5.11 DebugManagement.cpp File Reference

Debug management class source file.

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

Include dependency graph for DebugManagement.cpp:



Variables

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

5.11.1 Detailed Description

Debug management class source file.

Date

8 mai 2019

Author

nicls67

5.11.2 Variable Documentation

5.11.2.1 p_global_ASW_DebugManagement

`DebugManagement * p_global_ASW_DebugManagement`

Pointer to the `DebugManagement` object

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

5.11.2.2 str_debug_info_message_wdg_disabled

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

Info menu string displayed when the watchdog has been disabled.

Definition at line 94 of file DebugManagement.cpp.

5.11.2.3 str_debug_info_message_wdg_enabled

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

Info menu string displayed when the watchdog has been enabled.

Definition at line 99 of file DebugManagement.cpp.

5.11.2.4 str_debug_info_message_wdg_tmo_updated

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

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

Definition at line 84 of file DebugManagement.cpp.

5.11.2.5 str_debug_info_message_wdg_tmo_value

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

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

Definition at line 89 of file DebugManagement.cpp.

5.11.2.6 str_debug_info_message_wrong_menu_selection

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

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

Definition at line 79 of file DebugManagement.cpp.

5.11.2.7 str_debug_main_menu

```
const uint8_t str_debug_main_menu[ ]
```

Initial value:

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

Main menu of debug mode.

Definition at line 40 of file DebugManagement.cpp.

5.11.2.8 str_debug_wdg_menu

```
const uint8_t str_debug_wdg_menu[ ]
```

Initial value:

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

Watchdog menu of debug mode.

Definition at line 50 of file DebugManagement.cpp.

5.11.2.9 str_debug_wdg_timeout_update_selection

```
const uint8_t str_debug_wdg_timeout_update_selection[ ]
```

Initial value:

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

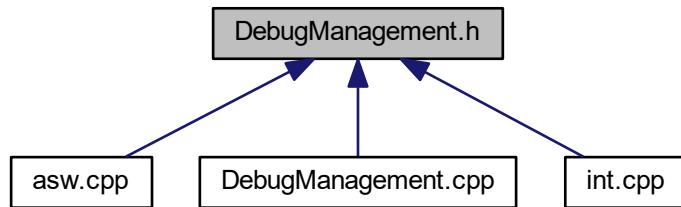
Watchdog timeout update selection.

Definition at line 61 of file DebugManagement.cpp.

5.12 DebugManagement.h File Reference

Debug management class header file.

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



Classes

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

Macros

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

Enumerations

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

Variables

- `DebugManagement * p_global_ASW_DebugManagement`

5.12.1 Detailed Description

Debug management class header file.

Date

8 mai 2019

Author

nicls67

5.12.2 Macro Definition Documentation

5.12.2.1 PERIOD_MS_TASK_DISPLAY_CPU_LOAD

```
#define PERIOD_MS_TASK_DISPLAY_CPU_LOAD 5000
```

Period for displaying CPU load data

Definition at line 14 of file DebugManagement.h.

5.12.2.2 PERIOD_MS_TASK_DISPLAY_DEBUG_DATA

```
#define PERIOD_MS_TASK_DISPLAY_DEBUG_DATA 5000
```

Period for displaying temperature and humidity data

Definition at line 13 of file DebugManagement.h.

5.12.3 Enumeration Type Documentation

5.12.3.1 debug_mgt_main_menu_state_t

```
enum debug_mgt_main_menu_state_t
```

Defines the debug states.

Enumerator

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

Definition at line 20 of file DebugManagement.h.

5.12.3.2 debug_mgt_wdg_state_t

```
enum debug_mgt_wdg_state_t
```

Defines possible states for watchdog management.

Enumerator

WDG_MAIN	Main menu of watchdog management
WDG_TMO_UPDATE	Timeout update mode

Definition at line 30 of file DebugManagement.h.

5.12.4 Variable Documentation

5.12.4.1 p_global_ASW_DebugManagement

`DebugManagement* p_global_ASW_DebugManagement`

Pointer to the `DebugManagement` object

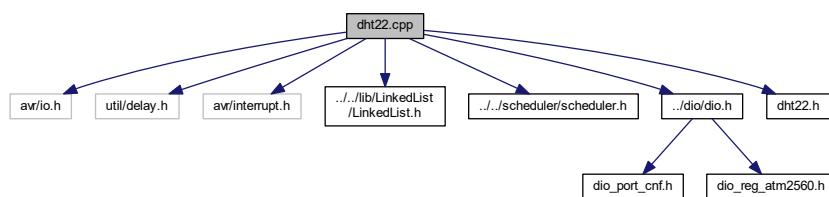
Definition at line 33 of file DebugManagement.cpp.

5.13 dht22.cpp File Reference

This file defines classes for DHT22 driver.

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

Include dependency graph for dht22.cpp:
```



Macros

- `#define MAX_WAIT_TIME_US 100`

Variables

- `dht22 * p_global_BSW_dht22`

5.13.1 Detailed Description

This file defines classes for DHT22 driver.

Date

23 mars 2018

Author

nicls67

5.13.2 Macro Definition Documentation

5.13.2.1 MAX_WAIT_TIME_US

```
#define MAX_WAIT_TIME_US 100
```

Maximum waiting time in microseconds

Definition at line 20 of file dht22.cpp.

5.13.3 Variable Documentation

5.13.3.1 p_global_BSW_dht22

```
dht22* p_global_BSW_dht22
```

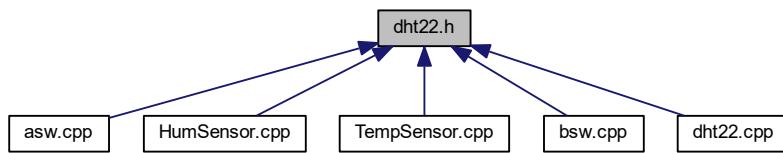
Pointer to `dht22` driver object

Definition at line 22 of file dht22.cpp.

5.14 dht22.h File Reference

DHT22 driver header file.

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



Classes

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

Variables

- [dht22 * p_global_BSW_dht22](#)

5.14.1 Detailed Description

DHT22 driver header file.

Date

23 mars 2018

Author

nicls67

5.14.2 Variable Documentation

5.14.2.1 [p_global_BSW_dht22](#)

`dht22* p_global_BSW_dht22`

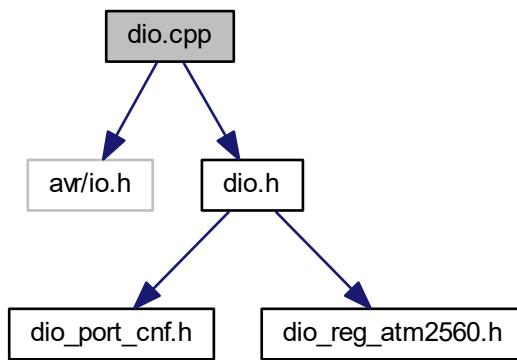
Pointer to [dht22](#) driver object

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

5.15 dio.cpp File Reference

DIO library.

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



Variables

- [dio * p_global_BSW_dio](#)

5.15.1 Detailed Description

DIO library.

Date

13 mars 2018

Author

nicls67

5.15.2 Variable Documentation

5.15.2.1 p_global_BSW_dio

`dio* p_global_BSW_dio`

Pointer to dio driver object

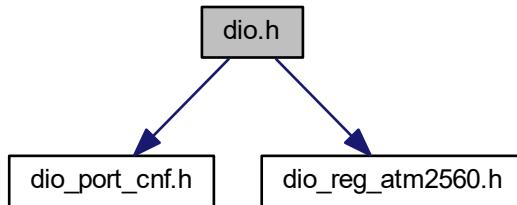
Definition at line 14 of file dio.cpp.

5.16 dio.h File Reference

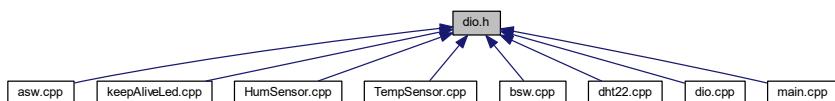
DIO library header file.

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

Include dependency graph for dio.h:



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



Classes

- class `dio`
DIO class.

Macros

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

Variables

- `dio * p_global_BSW_dio`

5.16.1 Detailed Description

DIO library header file.

Date

13 mars 2018

Author

nicls67

5.16.2 Macro Definition Documentation

5.16.2.1 DECODE_PIN

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

Macro used to extract pin index

Definition at line 20 of file dio.h.

5.16.2.2 DECODE_PORT

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

Macro used to extract port index

Definition at line 19 of file dio.h.

5.16.2.3 ENCODE_PORT

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

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

Definition at line 18 of file dio.h.

5.16.2.4 PORT_CNF_IN

```
#define PORT_CNF_IN 0
```

Pin is configured as input

Definition at line 16 of file dio.h.

5.16.2.5 PORT_CNF_OUT

```
#define PORT_CNF_OUT 1
```

Pin is configured as output

Definition at line 15 of file dio.h.

5.16.3 Variable Documentation

5.16.3.1 p_global_BSW_dio

```
dio* p_global_BSW_dio
```

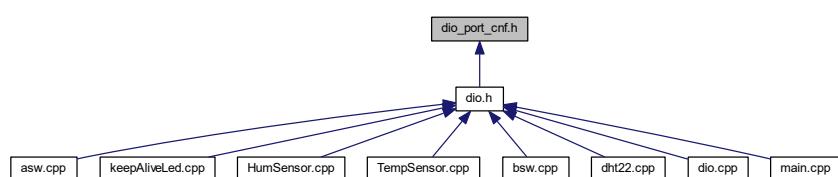
Pointer to dio driver object

Definition at line 14 of file dio.cpp.

5.17 dio_port_cnf.h File Reference

Digital ports configuration file.

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



Macros

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

5.17.1 Detailed Description

Digital ports configuration file.

Date

19 mars 2019

Author

nicls67

5.17.2 Macro Definition Documentation

5.17.2.1 PORT_A

`#define PORT_A 0`

PORTA index

Definition at line 42 of file dio_port_cnf.h.

5.17.2.2 PORT_B

`#define PORT_B 1`

PORTB index

Definition at line 43 of file dio_port_cnf.h.

5.17.2.3 PORT_C

```
#define PORT_C 2
```

PORTC index

Definition at line 44 of file dio_port_cnf.h.

5.17.2.4 PORT_D

```
#define PORT_D 3
```

PORTD index

Definition at line 45 of file dio_port_cnf.h.

5.17.2.5 PORTB_CNF_DDRC

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

Defines the configuration of DDRC register.

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

PB0 : N/A

PB1 : N/A

PB2 : N/A

PB3 : N/A

PB4 : IN

PB5 : N/A

PB6 : OUT

PB7 : OUT

Definition at line 25 of file dio_port_cnf.h.

5.17.2.6 PORTB_CNF_PORTB

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

Defines the configuration of PORTB register.

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

PB0 : N/A

PB1 : N/A

PB2 : N/A

PB3 : N/A

PB4 : Pull-up

PB5 : N/A

PB6 : HIGH

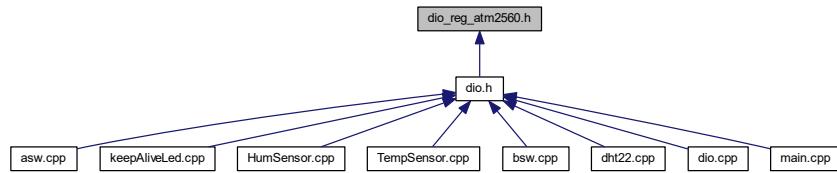
PB7 : LOW

Definition at line 40 of file dio_port_cnf.h.

5.18 dio_reg_atm2560.h File Reference

Defines DIO register addresses for ATMEGA2560.

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



Macros

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

5.18.1 Detailed Description

Defines DIO register addresses for ATMEGA2560.

Date

19 mars 2019

Author

nicls67

5.18.2 Macro Definition Documentation

5.18.2.1 DDRA_PTR

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

Macro defining pointer to DDR A register

Definition at line 24 of file dio_reg_atm2560.h.

5.18.2.2 DDRB_PTR

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

Macro defining pointer to DDR B register

Definition at line 25 of file dio_reg_atm2560.h.

5.18.2.3 DDRC_PTR

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

Macro defining pointer to DDR C register

Definition at line 26 of file dio_reg_atm2560.h.

5.18.2.4 DDRD_PTR

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

Macro defining pointer to DDR D register

Definition at line 27 of file dio_reg_atm2560.h.

5.18.2.5 PINA_PTR

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

Macro defining pointer to PIN A register

Definition at line 19 of file dio_reg_atm2560.h.

5.18.2.6 PINB_PTR

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

Macro defining pointer to PIN B register

Definition at line 20 of file dio_reg_atm2560.h.

5.18.2.7 PINC_PTR

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

Macro defining pointer to PIN C register

Definition at line 21 of file dio_reg_atm2560.h.

5.18.2.8 PIND_PTR

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

Macro defining pointer to PIN D register

Definition at line 22 of file dio_reg_atm2560.h.

5.18.2.9 PORTA_PTR

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

Macro defining pointer to PORT A register

Definition at line 14 of file dio_reg_atm2560.h.

5.18.2.10 PORTB_PTR

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

Macro defining pointer to PORT B register

Definition at line 15 of file dio_reg_atm2560.h.

5.18.2.11 PORTC_PTR

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

Macro defining pointer to PORT C register

Definition at line 16 of file dio_reg_atm2560.h.

5.18.2.12 PORTD_PTR

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

Macro defining pointer to PORT D register

Definition at line 17 of file dio_reg_atm2560.h.

5.19 DisplayInterface.cpp File Reference

Source code file for display services.

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

Include dependency graph for DisplayInterface.cpp:



Variables

- [DisplayInterface * p_global_ASW_DisplayInterface](#)

5.19.1 Detailed Description

Source code file for display services.

Date

23 avr. 2019

Author

nicls67

5.19.2 Variable Documentation

5.19.2.1 p_global_ASW_DisplayInterface

`DisplayInterface* p_global_ASW_DisplayInterface`

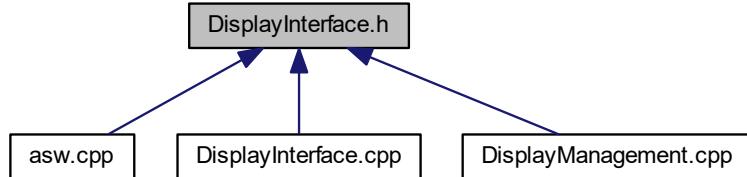
Pointer to `DisplayInterface` object

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

5.20 DisplayInterface.h File Reference

`DisplayInterface` class header file.

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



Classes

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

Macros

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

Enumerations

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

Variables

- `DisplayInterface * p_global_ASW_DisplayInterface`

5.20.1 Detailed Description

`DisplayInterface` class header file.

Date

23 avr. 2019

Author

nicls67

5.20.2 Macro Definition Documentation

5.20.2.1 DISPLAY_LINE_SHIFT_PERIOD_MS

```
#define DISPLAY_LINE_SHIFT_PERIOD_MS 500
```

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

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

5.20.2.2 DISPLAY_LINE_SHIFT_TEMPO_TIME

```
#define DISPLAY_LINE_SHIFT_TEMPO_TIME 6
```

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

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

5.20.3 Enumeration Type Documentation

5.20.3.1 T_DisplayInterface_LineAlignment

```
enum T_DisplayInterface_LineAlignment
```

Alignment mode for line display.

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

Enumerator

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

Definition at line 33 of file DisplayInterface.h.

5.20.3.2 T_DisplayInterface_LineDisplayMode

```
enum T_DisplayInterface_LineDisplayMode
```

Modes for line display.

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

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

Enumerator

NORMAL	
LINE_SHIFT	
GO_TO_NEXT_LINE	

Definition at line 20 of file DisplayInterface.h.

5.20.4 Variable Documentation

5.20.4.1 p_global_ASW_DisplayInterface

```
DisplayInterface* p_global_ASW_DisplayInterface
```

Pointer to [DisplayInterface](#) object

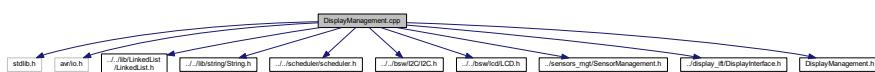
Definition at line 25 of file DisplayInterface.cpp.

5.21 DisplayManagement.cpp File Reference

Display management source file.

```
#include <stdlib.h>
#include <avr/io.h>
#include "../../lib/LinkedList/LinkedList.h"
#include "../../lib/string/String.h"
#include "../../scheduler/scheduler.h"
#include "../../bsw/I2C/I2C.h"
#include "../../bsw/lcd/LCD.h"
#include "../sensors_mgt/SensorManagement.h"
#include "../display_ift/DisplayInterface.h"
#include "DisplayManagement.h"
```

Include dependency graph for DisplayManagement.cpp:



Variables

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

5.21.1 Detailed Description

Display management source file.

Date

1 mai 2019

Author

nicls67

5.21.2 Variable Documentation

5.21.2.1 noSensorsDisplayString

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

String used in case sensors are deactivated

Definition at line 28 of file DisplayManagement.cpp.

5.21.2.2 p_global_ASW_DisplayManagement

`DisplayManagement* p_global_ASW_DisplayManagement`

Pointer to `DisplayManagement` object

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

5.21.2.3 welcomeMessageString

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

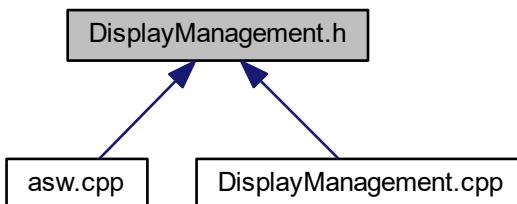
`String` displayed on the screen at startup

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

5.22 DisplayManagement.h File Reference

Display management class header file.

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



Classes

- class `DisplayManagement`

Display management class.

Macros

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

Variables

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

5.22.1 Detailed Description

Display management class header file.

Date

1 mai 2019

Author

nicls67

5.22.2 Macro Definition Documentation

5.22.2.1 DISPLAY_MGT_FIRST_LINE_SENSORS

```
#define DISPLAY_MGT_FIRST_LINE_SENSORS 0
```

Sensors data are displayed starting on line 0

Definition at line 18 of file DisplayManagement.h.

5.22.2.2 DISPLAY_MGT_I2C_BITRATE

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

I2C bus bitrate is 100 kHz

Definition at line 20 of file DisplayManagement.h.

5.22.2.3 DISPLAY_MGT_LCD_I2C_ADDR

```
#define DISPLAY_MGT_LCD_I2C_ADDR 0x27
```

I2C address of the screen

Definition at line 13 of file DisplayManagement.h.

5.22.2.4 DISPLAY_MGT_PERIOD_TASK_SENSOR

```
#define DISPLAY_MGT_PERIOD_TASK_SENSOR 1500
```

Display is updated every 1.5s

Definition at line 15 of file DisplayManagement.h.

5.22.2.5 DISPLAY_MGT_PERIOD_WELCOME_MSG_REMOVAL

```
#define DISPLAY_MGT_PERIOD_WELCOME_MSG_REMOVAL 5000
```

Time after which one the welcome message is removed

Definition at line 16 of file DisplayManagement.h.

5.22.3 Variable Documentation

5.22.3.1 LCD_init_cnf

```
const T_LCD_conf_struct LCD_init_cnf
```

Initial value:

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

LCD configuration structure.

This structure defines the initial configuration of the LCD screen.

Definition at line 26 of file DisplayManagement.h.

5.22.3.2 p_global_ASW_DisplayManagement

```
DisplayManagement* p_global_ASW_DisplayManagement
```

Pointer to [DisplayManagement](#) object

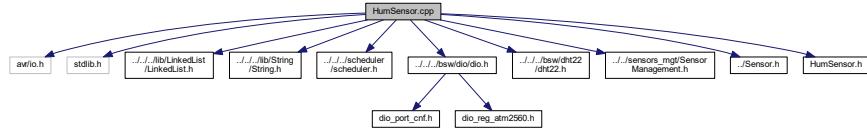
Definition at line 25 of file DisplayManagement.cpp.

5.23 HumSensor.cpp File Reference

Defines function of class [HumSensor](#).

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

Include dependency graph for HumSensor.cpp:



Macros

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

5.23.1 Detailed Description

Defines function of class [HumSensor](#).

Date

20 juin 2019

Author

nicls67

5.23.2 Macro Definition Documentation

5.23.2.1 DHT22_PORT

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

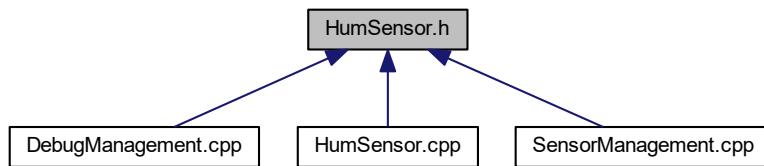
DHT22 is connected to port PB6

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

5.24 HumSensor.h File Reference

Class [HumSensor](#) header file.

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



Classes

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

5.24.1 Detailed Description

Class [HumSensor](#) header file.

Date

20 juin 2019

Author

nicls67

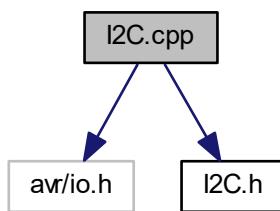
5.25 I2C.cpp File Reference

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

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

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

Include dependency graph for `I2C.cpp`:



Variables

- `I2C * p_global_BSW_i2c`

5.25.1 Detailed Description

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

Date

19 avr. 2019

Author

nicls67

5.25.2 Variable Documentation

5.25.2.1 `p_global_BSW_i2c`

`I2C* p_global_BSW_i2c`

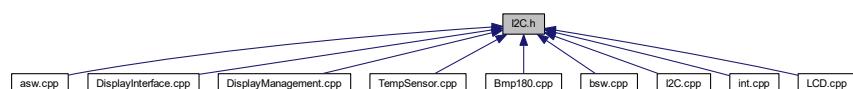
Pointer to [I2C](#) driver object

Definition at line 14 of file I2C.cpp.

5.26 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 REPEATED_START 0x10
- #define SLAW_ACK 0x18
- #define DATA_ACK 0x28
- #define SLAR_ACK 0x40

Variables

- I2C * p_global_BSW_i2c

5.26.1 Detailed Description

I2C class header file.

Date

19 avr. 2019

Author

nicls67

5.26.2 Macro Definition Documentation

5.26.2.1 DATA_ACK

```
#define DATA_ACK 0x28
```

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

Definition at line 16 of file I2C.h.

5.26.2.2 REPEATED_START

```
#define REPEATED_START 0x10
```

TWSR status code : REPEATED START condition transmitted

Definition at line 14 of file I2C.h.

5.26.2.3 SLAR_ACK

```
#define SLAR_ACK 0x40
```

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

Definition at line 17 of file I2C.h.

5.26.2.4 SLAW_ACK

```
#define SLAW_ACK 0x18
```

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

Definition at line 15 of file I2C.h.

5.26.2.5 START

```
#define START 0x08
```

TWSR status code : START condition transmitted

Definition at line 13 of file I2C.h.

5.26.3 Variable Documentation

5.26.3.1 p_global_BSW_i2c

```
I2C* p_global_BSW_i2c
```

Pointer to [I2C](#) driver object

Definition at line 14 of file I2C.cpp.

5.27 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 "../../../I2C/I2C.h"
#include "../../../bmp180/Bmp180.h"
#include "../../../asw/sensors_mgt/SensorManagement.h"
#include "../../../asw/debug_ift/DebugInterface.h"
#include "../../../asw/debug_mgt/DebugManagement.h"
#include "../../../asw/asw.h"
#include "../../../main.h"
```

Include dependency graph for int.cpp:



Functions

- **ISR (TIMER1_COMPA_vect)**
Main software interrupt.
- **ISR (TIMER3_COMPA_vect)**
Bmp180 end of conversion interrupt.
- **ISR (USART0_RX_vect)**
USART Rx Complete interrupt.

5.27.1 Detailed Description

Interrupt management source file.

Date

22 mai 2019

Author

nicls67

5.27.2 Function Documentation

5.27.2.1 ISR() [1/3]

```
ISR (
    TIMER1_COMPA_vect
)
```

Main software interrupt.

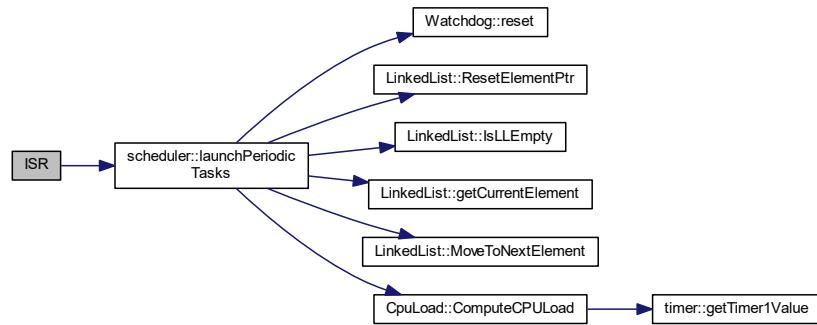
This function handles the interrupt raised by Timer #1. It wakes up the software every 500 ms to perform applications.

Returns

Nothing

Definition at line 36 of file int.cpp.

Here is the call graph for this function:



5.27.2.2 ISR() [2/3]

```
ISR (
    TIMER3_COMPA_vect
)
```

[Bmp180](#) end of conversion interrupt.

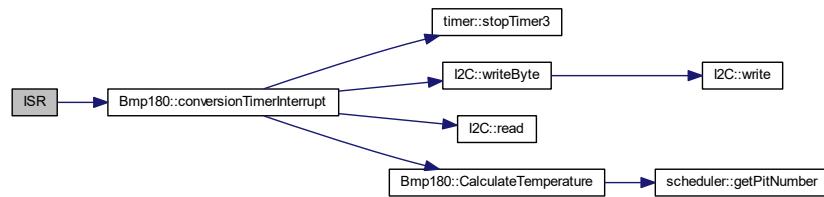
This function calls the end of conversion function of BMP180 driver.

Returns

Nothing

Definition at line 46 of file int.cpp.

Here is the call graph for this function:



5.27.2.3 ISR() [3/3]

```
ISR ( USART0_RX_vect )
```

USART Rx Complete interrupt.

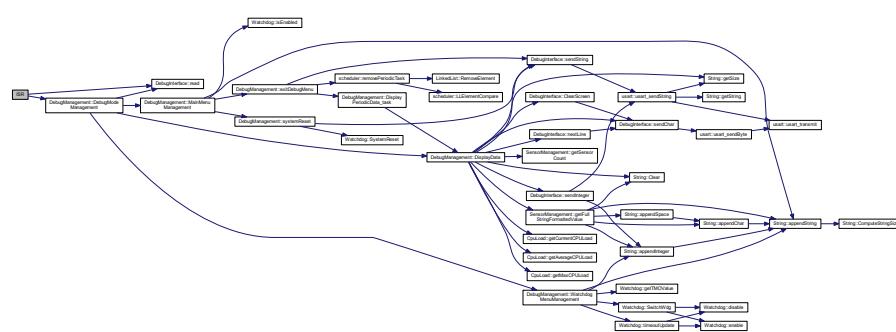
This function handles the interrupt raised when a frame has been received by USART. If debug mode mode is active, it calls debug mode management function. If inactive, it calls debug mode activation function if the received character is 'a'

Returns

Nothing

Definition at line 58 of file int.cpp.

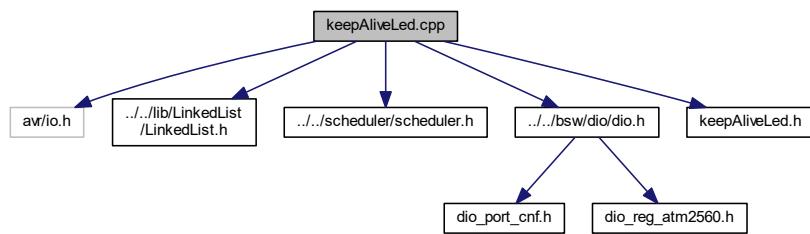
Here is the call graph for this function:



5.28 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](#)

5.28.1 Detailed Description

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

Date

17 mars 2018

Author

nicls67

5.28.2 Variable Documentation

5.28.2.1 `p_global_ASW_keepAliveLed`

`keepAliveLed* p_global_ASW_keepAliveLed`

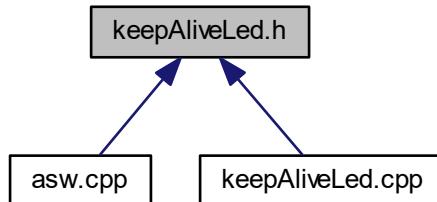
Pointer to [keepAliveLed](#) object

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

5.29 keepAliveLed.h File Reference

Class [keepAliveLed](#) header file.

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



Classes

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

Macros

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

Variables

- `keepAliveLed * p_global_ASW_keepAliveLed`

5.29.1 Detailed Description

Class [keepAliveLed](#) header file.

Date

17 mars 2018

Author

nicls67

5.29.2 Macro Definition Documentation

5.29.2.1 LED_PORT

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

LED is connected to port PB7

Definition at line 16 of file keepAliveLed.h.

5.29.2.2 PERIOD_MS_TASK_LED

```
#define PERIOD_MS_TASK_LED SW_PERIOD_MS
```

Period for led blinking

Definition at line 15 of file keepAliveLed.h.

5.29.3 Variable Documentation

5.29.3.1 p_global_ASW_keepAliveLed

```
keepAliveLed* p_global_ASW_keepAliveLed
```

Pointer to [keepAliveLed](#) object

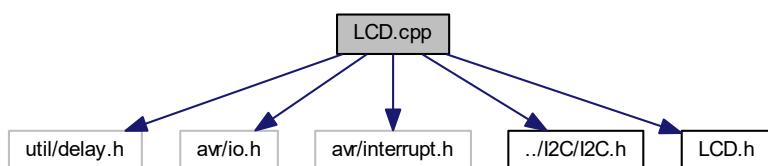
Definition at line 20 of file keepAliveLed.cpp.

5.30 LCD.cpp File Reference

[LCD](#) class source file.

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

Include dependency graph for LCD.cpp:



Variables

- `LCD * p_global_BSW_lcd`

5.30.1 Detailed Description

`LCD` class source file.

Date

20 avr. 2019

Author

nicls67

5.30.2 Variable Documentation

5.30.2.1 `p_global_BSW_lcd`

`LCD* p_global_BSW_lcd`

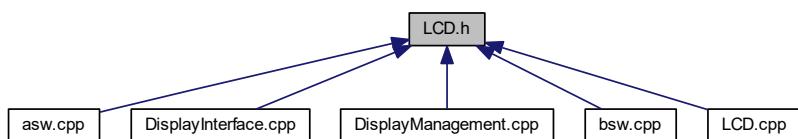
Pointer to `LCD` driver object

Definition at line 17 of file `LCD.cpp`.

5.31 LCD.h File Reference

`LCD` class header file.

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



Classes

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

Macros

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

Enumerations

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

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

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

Screen RAM definition.

Variables

- `LCD * p_global_BSW_lcd`

5.31.1 Detailed Description

`LCD` class header file.

Date

20 avr. 2019

Author

nicls67

5.31.2 Macro Definition Documentation

5.31.2.1 BACKLIGHT_PIN

```
#define BACKLIGHT_PIN 3
```

Backlight pin is on P3

Definition at line 17 of file LCD.h.

5.31.2.2 EN_PIN

```
#define EN_PIN 2
```

EN bit is on P2

Definition at line 14 of file LCD.h.

5.31.2.3 LCD_CNF_BACKLIGHT_OFF

```
#define LCD_CNF_BACKLIGHT_OFF 0
```

Backlight is disabled

Definition at line 70 of file LCD.h.

5.31.2.4 LCD_CNF_BACKLIGHT_ON

```
#define LCD_CNF_BACKLIGHT_ON 1
```

Backlight is enabled

Definition at line 69 of file LCD.h.

5.31.2.5 LCD_CNF_CURSOR_BLINK_OFF

```
#define LCD_CNF_CURSOR_BLINK_OFF 0
```

Cursor blinking is off, bit is set to 0

Definition at line 58 of file LCD.h.

5.31.2.6 LCD_CNF_CURSOR_BLINK_ON

```
#define LCD_CNF_CURSOR_BLINK_ON 1
```

Cursor blinking is on, bit is set to 1

Definition at line 57 of file LCD.h.

5.31.2.7 LCD_CNF_CURSOR_OFF

```
#define LCD_CNF_CURSOR_OFF 0
```

Cursor is off, bit is set to 0

Definition at line 54 of file LCD.h.

5.31.2.8 LCD_CNF_CURSOR_ON

```
#define LCD_CNF_CURSOR_ON 1
```

Cursor is on, bit is set to 1

Definition at line 53 of file LCD.h.

5.31.2.9 LCD_CNF_DISPLAY_OFF

```
#define LCD_CNF_DISPLAY_OFF 0
```

Display is off, bit is set to 0

Definition at line 50 of file LCD.h.

5.31.2.10 LCD_CNF_DISPLAY_ON

```
#define LCD_CNF_DISPLAY_ON 1
```

Display is on, bit is set to 1

Definition at line 49 of file LCD.h.

5.31.2.11 LCD_CNF_ENTRY_MODE_DIRECTION_LEFT

```
#define LCD_CNF_ENTRY_MODE_DIRECTION_LEFT 0
```

Direction of shift is left, bit is set to 0

Definition at line 62 of file LCD.h.

5.31.2.12 LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT

```
#define LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT 1
```

Direction of shift is right, bit is set to 1

Definition at line 61 of file LCD.h.

5.31.2.13 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF

```
#define LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF 0
```

Display shift is not performed, bit is set to 0

Definition at line 66 of file LCD.h.

5.31.2.14 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON

```
#define LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON 1
```

Display shift is performed, bit is set to 1

Definition at line 65 of file LCD.h.

5.31.2.15 LCD_CNF_FONT_5_11

```
#define LCD_CNF_FONT_5_11 1
```

Two-line configuration, bit is set to 1

Definition at line 46 of file LCD.h.

5.31.2.16 LCD_CNF_FONT_5_8

```
#define LCD_CNF_FONT_5_8 0
```

One-line configuration, bit is set to 0

Definition at line 45 of file LCD.h.

5.31.2.17 LCD_CNF_ONE_LINE

```
#define LCD_CNF_ONE_LINE 0
```

One-line configuration, bit is set to 0

Definition at line 41 of file LCD.h.

5.31.2.18 LCD_CNF_SHIFT_ID

```
#define LCD_CNF_SHIFT_ID 1
```

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

Definition at line 37 of file LCD.h.

5.31.2.19 LCD_CNF_SHIFT_SH

```
#define LCD_CNF_SHIFT_SH 0
```

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

Definition at line 38 of file LCD.h.

5.31.2.20 LCD_CNF_TWO_LINE

```
#define LCD_CNF_TWO_LINE 1
```

Two-line configuration, bit is set to 1

Definition at line 42 of file LCD.h.

5.31.2.21 LCD_DISPLAY_CTRL_FIELD_B

```
#define LCD_DISPLAY_CTRL_FIELD_B 0
```

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

Definition at line 34 of file LCD.h.

5.31.2.22 LCD_DISPLAY_CTRL_FIELD_C

```
#define LCD_DISPLAY_CTRL_FIELD_C 1
```

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

Definition at line 33 of file LCD.h.

5.31.2.23 LCD_DISPLAY_CTRL_FIELD_D

```
#define LCD_DISPLAY_CTRL_FIELD_D 2
```

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

Definition at line 32 of file LCD.h.

5.31.2.24 LCD_FCT_SET_FIELD_DL

```
#define LCD_FCT_SET_FIELD_DL 4
```

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

Definition at line 27 of file LCD.h.

5.31.2.25 LCD_FCT_SET_FIELD_F

```
#define LCD_FCT_SET_FIELD_F 2
```

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

Definition at line 29 of file LCD.h.

5.31.2.26 LCD_FCT_SET_FIELD_N

```
#define LCD_FCT_SET_FIELD_N 3
```

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

Definition at line 28 of file LCD.h.

5.31.2.27 LCD_INST_CLR_DISPLAY_BIT

```
#define LCD_INST_CLR_DISPLAY_BIT 0
```

Instruction bit for "clear display" is DB0

Definition at line 20 of file LCD.h.

5.31.2.28 LCD_INST_DISPLAY_CTRL

```
#define LCD_INST_DISPLAY_CTRL 3
```

Instruction bit for "display control" is DB3

Definition at line 22 of file LCD.h.

5.31.2.29 LCD_INST_ENTRY_MODE_SET

```
#define LCD_INST_ENTRY_MODE_SET 2
```

Instruction bit for "entry mode" is DB2

Definition at line 23 of file LCD.h.

5.31.2.30 LCD_INST_FUNCTION_SET

```
#define LCD_INST_FUNCTION_SET 5
```

Instruction bit for "function set" is DB5

Definition at line 21 of file LCD.h.

5.31.2.31 LCD_INST_SET_DDRAM_ADDR

```
#define LCD_INST_SET_DDRAM_ADDR 7
```

Instruction bit for "set DDRAM address" is DB7

Definition at line 24 of file LCD.h.

5.31.2.32 LCD_RAM_1_LINE_MAX

```
#define LCD_RAM_1_LINE_MAX 0x4F
```

Maximum address value in 1-line mode

Definition at line 74 of file LCD.h.

5.31.2.33 LCD_RAM_1_LINE_MIN

```
#define LCD_RAM_1_LINE_MIN 0
```

Minimum address value in 1-line mode

Definition at line 73 of file LCD.h.

5.31.2.34 LCD_RAM_2_LINES_MAX_1

```
#define LCD_RAM_2_LINES_MAX_1 0x27
```

Maximum address value in 2-lines mode for line 1

Definition at line 76 of file LCD.h.

5.31.2.35 LCD_RAM_2_LINES_MAX_2

```
#define LCD_RAM_2_LINES_MAX_2 0x67
```

Maximum address value in 2-lines mode for line 2

Definition at line 78 of file LCD.h.

5.31.2.36 LCD_RAM_2_LINES_MIN_1

```
#define LCD_RAM_2_LINES_MIN_1 0
```

Minimum address value in 2-lines mode for line 1

Definition at line 75 of file LCD.h.

5.31.2.37 LCD_RAM_2_LINES_MIN_2

```
#define LCD_RAM_2_LINES_MIN_2 0x40
```

Minimum address value in 2-lines mode for line 2

Definition at line 77 of file LCD.h.

5.31.2.38 LCD_SIZE_NB_CHAR_PER_LINE

```
#define LCD_SIZE_NB_CHAR_PER_LINE 20
```

LCD screen has 20 characters per line

Definition at line 85 of file LCD.h.

5.31.2.39 LCD_SIZE_NB_LINES

```
#define LCD_SIZE_NB_LINES 4
```

LCD screen has 4 lines

Definition at line 86 of file LCD.h.

5.31.2.40 LCD_WAIT_CLR_RETURN

```
#define LCD_WAIT_CLR_RETURN 1600
```

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

Definition at line 81 of file LCD.h.

5.31.2.41 LCD_WAIT_OTHER_MODES

```
#define LCD_WAIT_OTHER_MODES 40
```

Waiting time after all other modes is at least 38 us

Definition at line 82 of file LCD.h.

5.31.2.42 RS_PIN

```
#define RS_PIN 0
```

RS pin is on P0

Definition at line 16 of file LCD.h.

5.31.2.43 RW_PIN

```
#define RW_PIN 1
```

RW pin is on P1

Definition at line 15 of file LCD.h.

5.31.3 Enumeration Type Documentation

5.31.3.1 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.

5.31.3.2 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.

5.31.3.3 T_LCD_ram_area

```
enum T_LCD_ram_area
```

Screen RAM definition.

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

Enumerator

LCD_DATA_DDRAM	
LCD_DATA_CGRAM	

Definition at line 118 of file LCD.h.

5.31.4 Variable Documentation

5.31.4.1 p_global_BSW_lcd

`LCD* p_global_BSW_lcd`

Pointer to `LCD` driver object

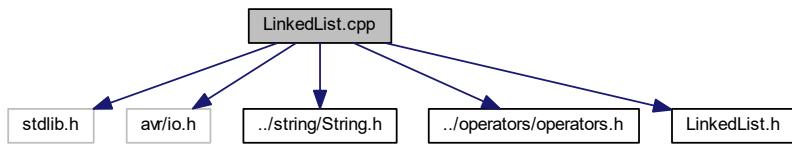
Definition at line 17 of file `LCD.cpp`.

5.32 LinkedList.cpp File Reference

Linked List library source file.

```
#include <stdlib.h>
#include <avr/io.h>
#include "../string/String.h"
#include "../operators/operators.h"
#include "LinkedList.h"
```

Include dependency graph for `LinkedList.cpp`:



5.32.1 Detailed Description

Linked List library source file.

Date

27 avr. 2019

Author

nicls67

5.33 LinkedList.h File Reference

Linked List library header file.

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



Classes

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

TypeDefs

- typedef [bool\(* CompareFctPtr_t\)](#) (`void *LLElement, void *CompareElement`)

5.33.1 Detailed Description

Linked List library header file.

Date

27 avr. 2019

Author

nicls67

5.33.2 Typedef Documentation

5.33.2.1 CompareFctPtr_t

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

Type defining a pointer to the comparison function

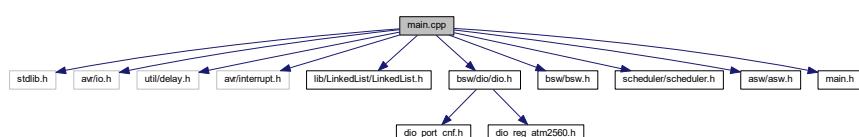
Definition at line 14 of file `LinkedList.h`.

5.34 main.cpp File Reference

Background task file.

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

Include dependency graph for `main.cpp`:



Macros

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

Functions

- `int main (void)`
Background task of program.

Variables

- `bool isDebugModeActivated`
- `const T_ASW_init_cnf ASW_init_cnf`

5.34.1 Detailed Description

Background task file.

Date

12 mars 2018

Author

nicls67

5.34.2 Macro Definition Documentation

5.34.2.1 DEBUG_ACTIVE_PORT

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

Debug activation pin is port PB6

Definition at line 26 of file main.cpp.

5.34.3 Function Documentation

5.34.3.1 main()

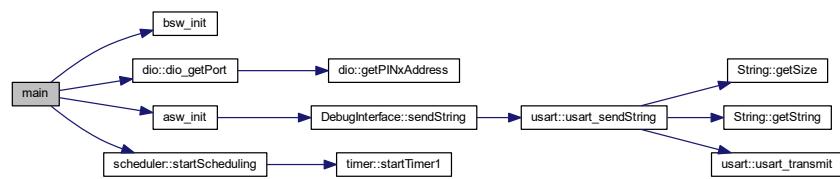
```
int main (
    void )
```

Background task of program.

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

Definition at line 45 of file main.cpp.

Here is the call graph for this function:



5.34.4 Variable Documentation

5.34.4.1 ASW_init_cnf

```
const T_ASW_init_cnf ASW_init_cnf
```

Initial value:

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

Definition of needed ASW services

Definition at line 30 of file main.cpp.

5.34.4.2 isDebugEnabled

```
bool isDebugEnabled
```

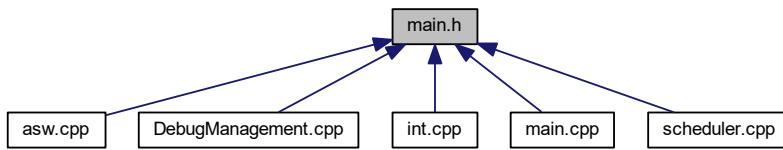
Flag indicating if the debug mode is activated or not

Definition at line 28 of file main.cpp.

5.35 main.h File Reference

Background task header file.

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



Variables

- bool `isDebugModeActivated`
- const `T_ASW_init_cnf ASW_init_cnf`

5.35.1 Detailed Description

Background task header file.

Date

17 mars 2018

Author

nicls67

5.35.2 Variable Documentation

5.35.2.1 ASW_init_cnf

```
const T_ASW_init_cnf ASW_init_cnf
```

Definition of needed ASW services

Definition at line 30 of file main.cpp.

5.35.2.2 isDebugEnabledActivated

```
bool isDebugEnabledActivated
```

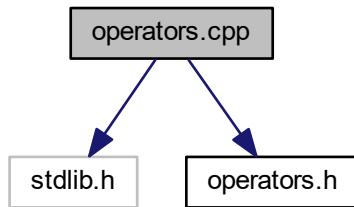
Flag indicating if the debug mode is activated or not

Definition at line 28 of file main.cpp.

5.36 operators.cpp File Reference

c++ operators definitions

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



Functions

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

5.36.1 Detailed Description

c++ operators definitions

Date

14 mars 2018

Author

nicls67

5.36.2 Function Documentation

5.36.2.1 operator delete()

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

Operator delete.

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

Parameters

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

Returns

Nothing

Definition at line 18 of file operators.cpp.

5.36.2.2 operator new()

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

Operator new.

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

Parameters

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

Returns

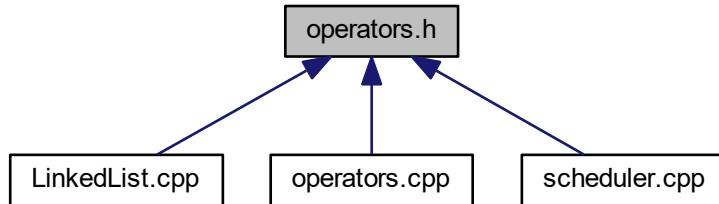
Pointer to the start of allocated memory zone

Definition at line 13 of file operators.cpp.

5.37 operators.h File Reference

C++ operators definitions header file

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



Functions

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

5.37.1 Detailed Description

c++ operators definitions header file

Date

14 mars 2018

Author

nicls67

5.37.2 Function Documentation

5.37.2.1 operator delete()

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

Operator delete.

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

Parameters

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

Returns

Nothing

Definition at line 18 of file operators.cpp.

5.37.2.2 operator new()

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

Operator new.

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

Parameters

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

Returns

Pointer to the start of allocated memory zone

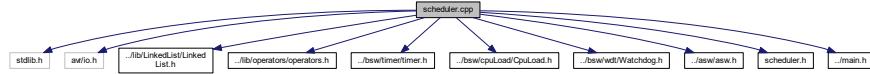
Definition at line 13 of file operators.cpp.

5.38 scheduler.cpp File Reference

Defines scheduler class.

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

Include dependency graph for scheduler.cpp:



Variables

- `scheduler * p_global_scheduler`

5.38.1 Detailed Description

Defines scheduler class.

Date

16 mars 2018

Author

nicls67

5.38.2 Variable Documentation

5.38.2.1 `p_global_scheduler`

`scheduler* p_global_scheduler`

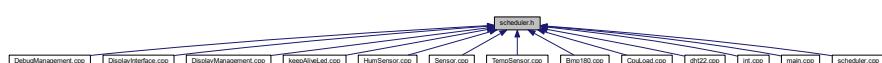
Pointer to scheduler object

Definition at line 27 of file scheduler.cpp.

5.39 scheduler.h File Reference

Scheduler class header file.

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



Classes

- class `scheduler`
Scheduler class.
- struct `scheduler::Task_t`
Type defining a task structure.

Macros

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

Typedefs

- `typedef void(* TaskPtr_t) (void)`
Type defining a pointer to function.

Variables

- `scheduler * p_global_scheduler`

5.39.1 Detailed Description

Scheduler class header file.

Date

16 mars 2018

Author

nicls67

5.39.2 Macro Definition Documentation

5.39.2.1 PRESCALER_PERIODIC_TIMER

```
#define PRESCALER_PERIODIC_TIMER 256
```

Value of prescaler to use for periodic timer

Definition at line 16 of file scheduler.h.

5.39.2.2 SW_PERIOD_MS

```
#define SW_PERIOD_MS 500
```

Software period, used to define periodic timer interrupt

Definition at line 15 of file scheduler.h.

5.39.2.3 TIMER_CTC_VALUE

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

Compare value for periodic timer

Definition at line 17 of file scheduler.h.

5.39.3 Typedef Documentation

5.39.3.1 TaskPtr_t

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

Type defining a pointer to function.

Definition at line 22 of file scheduler.h.

5.39.4 Variable Documentation

5.39.4.1 p_global_scheduler

```
scheduler* p_global_scheduler
```

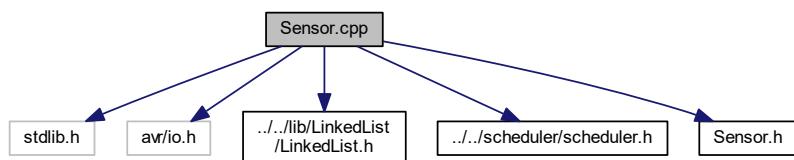
Pointer to scheduler object

Definition at line 27 of file scheduler.cpp.

5.40 Sensor.cpp File Reference

[Sensor](#) class source code file.

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



Macros

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

5.40.1 Detailed Description

[Sensor](#) class source code file.

Date

20 juin 2019

Author

nicls67

5.40.2 Macro Definition Documentation

5.40.2.1 TASK_PERIOD_DEFAULT

```
#define TASK_PERIOD_DEFAULT 1000
```

Default sensor task period : 1s

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

5.40.2.2 VALIDITY_TIMEOUT_MS_DEFAULT

```
#define VALIDITY_TIMEOUT_MS_DEFAULT 30000
```

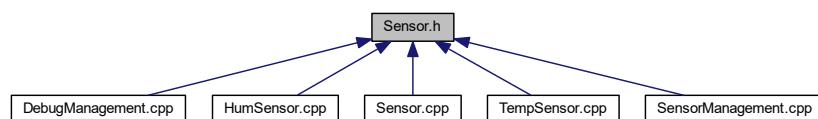
[Sensor](#) data are declared invalid after 30s

Definition at line 19 of file [Sensor.cpp](#).

5.41 Sensor.h File Reference

[Sensor](#) class header file.

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



Classes

- class [Sensor](#)

Generic class for sensor device.

5.41.1 Detailed Description

[Sensor](#) class header file.

Date

20 juin 2019

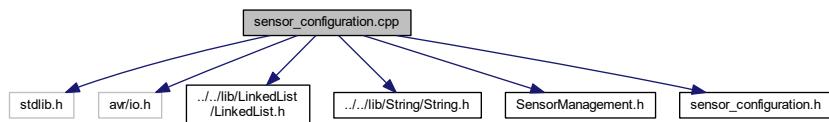
Author

nicls67

5.42 sensor_configuration.cpp File Reference

[Sensor](#) configuration file.

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



Macros

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

Variables

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

5.42.1 Detailed Description

Sensor configuration file.

Date

22 juin 2019

Author

nicls67

5.42.2 Macro Definition Documentation

5.42.2.1 SENSOR_MGT_CNF_DEFAULT_PERIOD

```
#define SENSOR_MGT_CNF_DEFAULT_PERIOD 1000
```

Default period for sensors task

Definition at line 19 of file sensor_configuration.cpp.

5.42.2.2 SENSOR_MGT_CNF_DEFAULT_TMO

```
#define SENSOR_MGT_CNF_DEFAULT_TMO 15000
```

Default timeout value for sensors

Definition at line 20 of file sensor_configuration.cpp.

5.42.3 Variable Documentation

5.42.3.1 SensorManagement_Sensor_Config_list

```
T_SensorManagement_Sensor_Config SensorManagement_Sensor_Config_list[2]
```

Initial value:

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

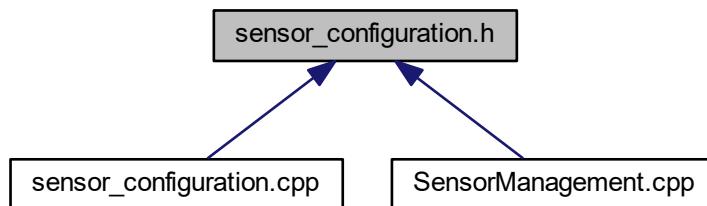
[Sensor](#) configuration table.

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

5.43 sensor_configuration.h File Reference

Sensors configuration header file.

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



Classes

- struct [T_SensorManagement_Sensor_Config](#)
Sensor informations structure.

Variables

- [T_SensorManagement_Sensor_Config](#) `SensorManagement_Sensor_Config_list [2]`
Sensor configuration table.

5.43.1 Detailed Description

Sensors configuration header file.

Date

22 juin 2019

Author

nicls67

5.43.2 Variable Documentation

5.43.2.1 SensorManagement_Sensor_Config_list

`T_SensorManagement_Sensor_Config SensorManagement_Sensor_Config_list[2]`

Sensor configuration table.

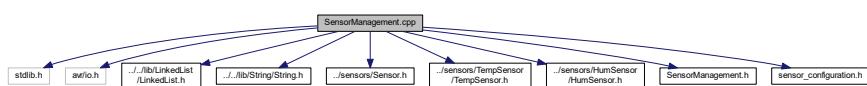
Definition at line 25 of file `sensor_configuration.cpp`.

5.44 SensorManagement.cpp File Reference

[SensorManagement](#) class source code file.

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

Include dependency graph for `SensorManagement.cpp`:



Variables

- `SensorManagement * p_global_ASW_SensorManagement`

5.44.1 Detailed Description

[SensorManagement](#) class source code file.

Date

22 juin 2019

Author

nicls67

5.44.2 Variable Documentation

5.44.2.1 p_global_ASW_SensorManagement

[SensorManagement](#)* p_global_ASW_SensorManagement

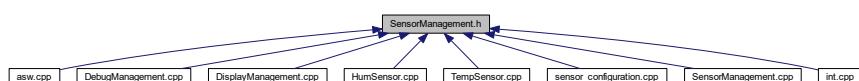
Pointer to the [SensorManagement](#) object

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

5.45 SensorManagement.h File Reference

[SensorManagement](#) class header file.

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



Classes

- class [SensorManagement](#)

Sensor management class.

Enumerations

- enum [T_SensorManagement_Sensor_Type](#) { [TEMPERATURE](#), [HUMIDITY](#) }

Sensor type enumeration.

Variables

- `SensorManagement * p_global_ASW_SensorManagement`

5.45.1 Detailed Description

`SensorManagement` class header file.

Date

22 juin 2019

Author

nicls67

5.45.2 Enumeration Type Documentation

5.45.2.1 T_SensorManagement_Sensor_Type

`enum T_SensorManagement_Sensor_Type`

`Sensor` type enumeration.

This enumeration defines all types of sensors available.

Enumerator

TEMPERATURE	
HUMIDITY	

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

5.45.3 Variable Documentation

5.45.3.1 p_global_ASW_SensorManagement

`SensorManagement* p_global_ASW_SensorManagement`

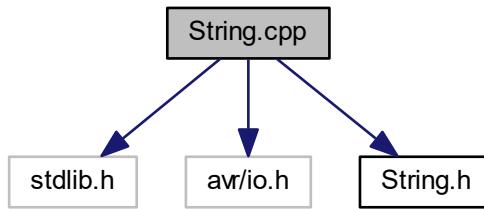
Pointer to the `SensorManagement` object

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

5.46 String.cpp File Reference

[String](#) class source file.

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



5.46.1 Detailed Description

[String](#) class source file.

Date

2 mai 2019

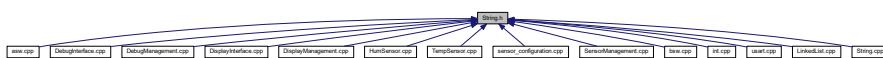
Author

nicls67

5.47 String.h File Reference

[String](#) class header file.

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



Classes

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

5.47.1 Detailed Description

[String](#) class header file.

Date

2 mai 2019

Author

nicls67

5.48 TempSensor.cpp File Reference

Defines function of class [TempSensor](#).

```
#include <avr/io.h>
#include <stdlib.h>
#include "../../lib/LinkedList/LinkedList.h"
#include "../../lib/String/String.h"
#include "../../scheduler/scheduler.h"
#include "../../bsw/usart/usart.h"
#include "../../bsw/dio/dio.h"
#include "../../bsw/dht22/dht22.h"
#include "../../bsw/I2C/I2C.h"
#include "../../bsw/bmp180/Bmp180.h"
#include "../../debug_ift/DebugInterface.h"
#include "../../sensors_mgt/SensorManagement.h"
#include "../Sensor.h"
#include "TempSensor.h"
```

Include dependency graph for TempSensor.cpp:



Macros

- `#define DHT22_PORT ENCODE_PORT(PORT_B, 6)`
- `#define TEMP_SENSOR_VALUE_ACCEPTABLE_DIFFERENCE 10`

5.48.1 Detailed Description

Defines function of class [TempSensor](#).

Date

23 mars 2018

Author

nicls67

5.48.2 Macro Definition Documentation

5.48.2.1 DHT22_PORT

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

DHT22 is connected to port PB6

Definition at line 28 of file TempSensor.cpp.

5.48.2.2 TEMP_SENSOR_VALUE_ACCEPTABLE_DIFFERENCE

```
#define TEMP_SENSOR_VALUE_ACCEPTABLE_DIFFERENCE 10
```

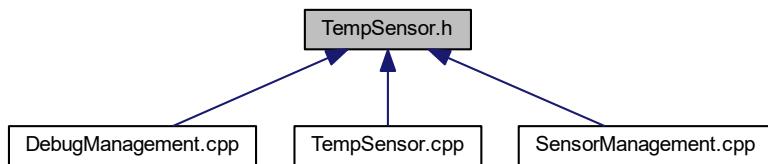
Sensors value can have a difference of 1 degree before becoming invalid

Definition at line 29 of file TempSensor.cpp.

5.49 TempSensor.h File Reference

Class [TempSensor](#) header file.

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



Classes

- class [TempSensor](#)

Class for temperature sensor.

5.49.1 Detailed Description

Class [TempSensor](#) header file.

Date

23 mars 2018

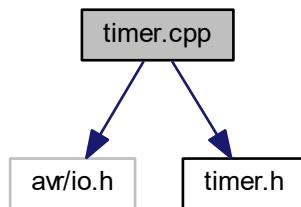
Author

nicls67

5.50 timer.cpp File Reference

Defines function for class timer.

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



Variables

- [timer * p_global_BSW_timer](#)

5.50.1 Detailed Description

Defines function for class timer.

Date

15 mars 2018

Author

nicls67

5.50.2 Variable Documentation

5.50.2.1 p_global_BSW_timer

`timer* p_global_BSW_timer`

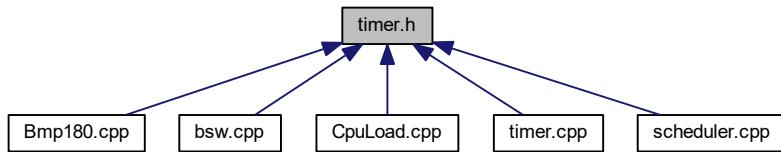
Pointer to timer driver object

Definition at line 13 of file timer.cpp.

5.51 timer.h File Reference

Timer class header file.

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



Classes

- class `timer`
Class defining a timer.

Variables

- `timer * p_global_BSW_timer`

5.51.1 Detailed Description

Timer class header file.

Date

15 mars 2018

Author

nicls67

5.51.2 Variable Documentation

5.51.2.1 p_global_BSW_timer

```
timer* p_global_BSW_timer
```

Pointer to timer driver object

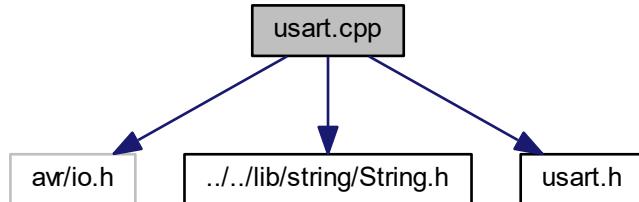
Definition at line 13 of file timer.cpp.

5.52 usart.cpp File Reference

BSW library for USART.

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

Include dependency graph for usart.cpp:



Variables

- `uart * p_global_BSW_usart`

5.52.1 Detailed Description

BSW library for USART.

Date

13 mars 2018

Author

nicls67

5.52.2 Variable Documentation

5.52.2.1 p_global_BSW_usart

`usart*` `p_global_BSW_usart`

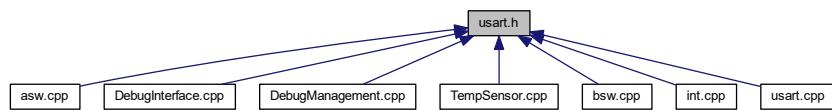
Pointer to usart driver object

Definition at line 16 of file usart.cpp.

5.53 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`

5.53.1 Detailed Description

Header file for USART library.

Date

13 mars 2018

Author

nicls67

5.53.2 Variable Documentation

5.53.2.1 p_global_BSW_usart

```
usart* p_global_BSW_usart
```

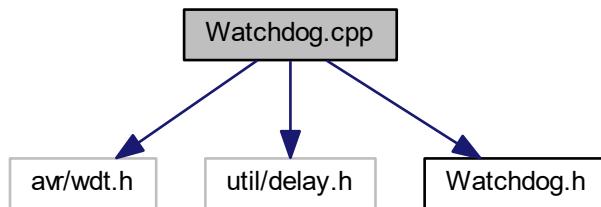
Pointer to usart driver object

Definition at line 16 of file usart.cpp.

5.54 Watchdog.cpp File Reference

Class [Watchdog](#) source code file.

```
#include <avr/wdt.h>
#include <util/delay.h>
#include "Watchdog.h"
Include dependency graph for Watchdog.cpp:
```



Macros

- [#define WDG_TIMEOUT_DEFAULT_MS WDG_TMO_500MS](#)

Variables

- [Watchdog * p_global_BSW_wdg](#)

5.54.1 Detailed Description

Class [Watchdog](#) source code file.

Date

6 juin 2019

Author

nicls67

5.54.2 Macro Definition Documentation

5.54.2.1 WDG_TIMEOUT_DEFAULT_MS

```
#define WDG_TIMEOUT_DEFAULT_MS WDG_TMO_500MS
```

Default timeout value is set to 500 ms

Definition at line 15 of file Watchdog.cpp.

5.54.3 Variable Documentation

5.54.3.1 p_global_BSW_wdg

```
Watchdog* p_global_BSW_wdg
```

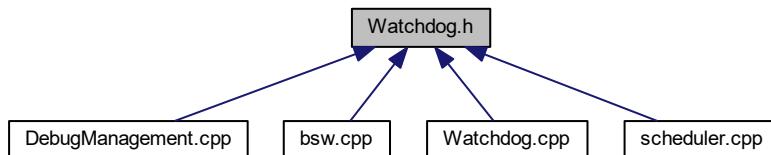
Pointer to [Watchdog](#) driver object

Definition at line 17 of file Watchdog.cpp.

5.55 Watchdog.h File Reference

Class [Watchdog](#) header file.

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



Classes

- class [Watchdog](#)
Watchdog management class.

Macros

- `#define WDG_TMO_15MS WDTO_15MS`
Definition of available timeout values.
- `#define WDG_TMO_30MS WDTO_30MS`
- `#define WDG_TMO_60MS WDTO_60MS`
- `#define WDG_TMO_120MS WDTO_120MS`
- `#define WDG_TMO_250MS WDTO_250MS`
- `#define WDG_TMO_500MS WDTO_500MS`
- `#define WDG_TMO_1S WDTO_1S`
- `#define WDG_TMO_2S WDTO_2S`
- `#define WDG_TMO_4S WDTO_4S`
- `#define WDG_TMO_8S WDTO_8S`

Variables

- [Watchdog * p_global_BSW_wdg](#)

5.55.1 Detailed Description

Class [Watchdog](#) header file.

Date

6 juin 2019

Author

nicls67

5.55.2 Macro Definition Documentation

5.55.2.1 WDG_TMO_120MS

```
#define WDG_TMO_120MS WDTO_120MS
```

Timeout value is 120 ms

Definition at line 19 of file Watchdog.h.

5.55.2.2 WDG_TMO_15MS

```
#define WDG_TMO_15MS WDTO_15MS
```

Definition of available timeout values.

Timeout value is 15 ms

Definition at line 16 of file Watchdog.h.

5.55.2.3 WDG_TMO_1S

```
#define WDG_TMO_1S WDTO_1S
```

Timeout value is 1 s

Definition at line 22 of file Watchdog.h.

5.55.2.4 WDG_TMO_250MS

```
#define WDG_TMO_250MS WDTO_250MS
```

Timeout value is 250 ms

Definition at line 20 of file Watchdog.h.

5.55.2.5 WDG_TMO_2S

```
#define WDG_TMO_2S WDTO_2S
```

Timeout value is 2 s

Definition at line 23 of file Watchdog.h.

5.55.2.6 WDG_TMO_30MS

```
#define WDG_TMO_30MS WDTO_30MS
```

Timeout value is 30 ms

Definition at line 17 of file Watchdog.h.

5.55.2.7 WDG_TMO_4S

```
#define WDG_TMO_4S WDTO_4S
```

Timeout value is 4 s

Definition at line 24 of file Watchdog.h.

5.55.2.8 WDG_TMO_500MS

```
#define WDG_TMO_500MS WDTO_500MS
```

Timeout value is 500 ms

Definition at line 21 of file Watchdog.h.

5.55.2.9 WDG_TMO_60MS

```
#define WDG_TMO_60MS WDTO_60MS
```

Timeout value is 60 ms

Definition at line 18 of file Watchdog.h.

5.55.2.10 WDG_TMO_8S

```
#define WDG_TMO_8S WDTO_8S
```

Timeout value is 8 s

Definition at line 25 of file Watchdog.h.

5.55.3 Variable Documentation

5.55.3.1 p_global_BSW_wdg

`Watchdog*` `p_global_BSW_wdg`

Pointer to `Watchdog` driver object

Definition at line 17 of file Watchdog.cpp.

Index

~LinkedList
 LinkedList, 110

~String
 String, 140

AC1
 Bmp180::T_BMP180_calib_data, 150

AC2
 Bmp180::T_BMP180_calib_data, 150

AC3
 Bmp180::T_BMP180_calib_data, 150

AC4
 Bmp180::T_BMP180_calib_data, 150

AC5
 Bmp180::T_BMP180_calib_data, 151

AC6
 Bmp180::T_BMP180_calib_data, 151

ASW_init_cnf
 main.cpp, 256
 main.h, 257

addPeriodicTask
 scheduler, 117

alignment
 T_display_data, 154

appendBool
 String, 141

appendChar
 String, 141

appendInteger
 String, 142

appendSpace
 String, 143

appendString
 String, 144

asw.cpp, 187
 asw_init, 188

asw.h, 189
 asw_init, 189

asw_init
 asw.cpp, 188
 asw.h, 189

AttachNewElement
 LinkedList, 111

avg_load
 CpuLoad, 23

B1
 Bmp180::T_BMP180_calib_data, 151

B2
 Bmp180::T_BMP180_calib_data, 151

BACKLIGHT_PIN
 LCD.h, 243

BMP180_CHIP_ID_CALIB_EEP_START_ADDR
 Bmp180.h, 193

BMP180_CHIP_ID_EEP_ADDR
 Bmp180.h, 193

BMP180_CHIP_ID_EXPECTED
 Bmp180.h, 193

BMP180_CTRL_MEAS_EEP_ADDR
 Bmp180.h, 193

BMP180_CTRL_MEAS_START_TEMP_CONV
 Bmp180.h, 194

BMP180_I2C_ADDR
 Bmp180.h, 194

BMP180_I2C_BITRATE
 Bmp180.h, 194

BMP180_MONITORING_DEFAULT_PERIOD
 Bmp180.h, 194

BMP180_OUT_REG_LSB EEPROM_ADDR
 Bmp180.h, 194

BMP180_OUT_REG_MSB EEPROM_ADDR
 Bmp180.h, 195

BMP180_TEMP_MEAS_TIMER_CTC_VALUE
 Bmp180.h, 195

BMP180_TIMER_PRESCALER_VALUE
 Bmp180.h, 195

backlight_en
 T_LCD_conf_struct, 157

backlight_enable
 LCD, 106

BaudRate
 uart, 177

bitrate
 I2C, 89

blinkLed_task
 keepAliveLed, 90

Bmp180, 9
 Bmp180, 10
 Bmp180Monitoring_Task, 11
 CalculateTemperature, 12
 calibration_data, 18
 chip_id, 19
 conversionTimerInterrupt, 13
 getMonitoringTaskPeriod, 14
 getStatus, 14
 getTemperatureValue, 14
 i2c_drv_ptr, 19
 pressure_value, 19
 readCalibData, 15

readChipID, 16
 startNewTemperatureConversion, 17
 status, 19
 task_period, 19
 temperature_value, 20
 TemperatureMonitoring, 18
 Bmp180.cpp, 190
 p_global_BSW_bmp180, 191
 Bmp180.h, 192
 BMP180_CHIP_ID_CALIB_EEP_START_ADDR, 193
 BMP180_CHIP_ID_EEP_ADDR, 193
 BMP180_CHIP_ID_EXPECTED, 193
 BMP180_CTRL_MEAS_EEP_ADDR, 193
 BMP180_CTRL_MEAS_START_TEMP_CONV, 194
 BMP180_I2C_ADDR, 194
 BMP180_I2C_BITRATE, 194
 BMP180_MONITORING_DEFAULT_PERIOD, 194
 BMP180_OUT_REG_LSB EEPROM_ADDR, 194
 BMP180_OUT_REG_MSB EEPROM_ADDR, 195
 BMP180_TEMP_MEAS_TIMER_CTC_VALUE, 195
 BMP180_TIMER_PRESCALER_VALUE, 195
 p_global_BSW_bmp180, 196
 T_BMP180_status, 195
 Bmp180::T_BMP180_calib_data, 149
 AC1, 150
 AC2, 150
 AC3, 150
 AC4, 150
 AC5, 151
 AC6, 151
 B1, 151
 B2, 151
 MB, 151
 MC, 151
 MD, 152
 Bmp180::T_BMP180_measurement_data, 152
 ready, 152
 ts, 152
 value, 153
 Bmp180Monitoring_Task
 Bmp180, 11
 bsw.cpp, 196
 bsw_init, 197
 bsw.h, 198
 bsw_init, 198
 bsw_init
 bsw.cpp, 197
 bsw.h, 198
 CalculateTemperature
 Bmp180, 12
 calibration_data
 Bmp180, 18
 chip_id
 Bmp180, 19
 Clear
 String, 145
 ClearFullScreen
 DisplayInterface, 64
 ClearLine
 DisplayInterface, 65
 ClearScreen
 DebugInterface, 28
 ClearStringInDataStruct
 DisplayInterface, 66
 cnfCursorBlink
 LCD, 106
 cnfCursorOnOff
 LCD, 106
 cnfDisplayOnOff
 LCD, 106
 cnfEntryModeDir
 LCD, 106
 cnfEntryModeShift
 LCD, 106
 cnfFontType
 LCD, 107
 cnfI2C_addr
 LCD, 107
 cnfLineNumber
 LCD, 107
 command
 LCD, 94
 CompareFctPtr_t
 LinkedList.h, 254
 ComputeCPUload
 CpuLoad, 21
 ComputeStringSize
 String, 145
 ConfigureBacklight
 LCD, 95
 ConfigureCursorBlink
 LCD, 95
 ConfigureCursorOnOff
 LCD, 96
 ConfigureDisplayOnOff
 LCD, 97
 ConfigureEntryModeDir
 LCD, 97
 ConfigureEntryModeShift
 LCD, 98
 ConfigureFontType
 LCD, 99
 ConfigureI2CAddr
 LCD, 99
 ConfigureLineNumber
 LCD, 100
 configureTimer1
 timer, 168
 configureTimer3
 timer, 169
 conversionTimerInterrupt
 Bmp180, 13

CpuLoad, 20
avg_load, 23
ComputeCPULoad, 21
CpuLoad, 21
current_load, 24
getAverageCPULoad, 22
getCurrentCPULoad, 22
getMaxCPULoad, 23
last_sum_value, 24
max_load, 24
sample_cnt, 24
sample_idx, 24
sample_mem, 25
CpuLoad.cpp, 199
p_global_BSW_cpupload, 200
CpuLoad.h, 200
NB_OF_SAMPLES, 201
p_global_BSW_cpupload, 201
curElement_ptr
 LinkedList, 115
current_load
 CpuLoad, 24
cursor_en
 T_LCD_conf_struct, 157
cursorBlink_en
 T_LCD_conf_struct, 157

DATA_ACK
 I2C.h, 233
DDRA_PTR
 dio_reg_atm2560.h, 219
DDRB_PTR
 dio_reg_atm2560.h, 220
DDRC_PTR
 dio_reg_atm2560.h, 220
DDRD_PTR
 dio_reg_atm2560.h, 220
DEBUG_ACTIVE_PORT
 main.cpp, 255
DECODE_PIN
 dio.h, 215
DECODE_PORT
 dio.h, 215
DHT22_PORT
 HumSensor.cpp, 230
 TempSensor.cpp, 274
DISPLAY_LINE_SHIFT_PERIOD_MS
 DisplayInterface.h, 224
DISPLAY_LINE_SHIFT_TEMPO_TIME
 DisplayInterface.h, 224
DISPLAY_MGT_FIRST_LINE_SENSORS
 DisplayManagement.h, 228
DISPLAY_MGT_I2C_BITRATE
 DisplayManagement.h, 228
DISPLAY_MGT_LCD_I2C_ADDR
 DisplayManagement.h, 228
DISPLAY_MGT_PERIOD_TASK_SENSOR
 DisplayManagement.h, 228
DISPLAY_MGT_PERIOD_WELCOME_MSG_REMOTE
 VAL
 DisplayManagement.h, 229
 data_name_str
 T_SensorManagement_Sensor_Config, 161
 data_ptr
 LinkedList::T_LL_element, 160
ddram_addr
 LCD, 107
debug_ift_ptr
 DebugManagement, 44
debug_mgt_main_menu_state_t
 DebugManagement.h, 209
debug_mgt_state_struct_t, 25
 main_state, 25
 wdg_state, 26
debug_mgt_wdg_state_t
 DebugManagement.h, 209
debug_state
 DebugManagement, 44
DebugInterface, 26
 ClearScreen, 28
 DebugInterface, 27
 nextLine, 28
 read, 29
 sendBool, 29
 sendChar, 30
 sendInteger, 31
 sendString, 32, 33
 usart_drv_ptr, 34
DebugInterface.cpp, 202
 p_global_ASW_DebugInterface, 202
DebugInterface.h, 203
 p_global_ASW_DebugInterface, 204
 USART_BAUDRATE, 203
DebugManagement, 34
 debug_ift_ptr, 44
 debug_state, 44
 DebugManagement, 36
 DebugModeManagement, 36
 DisplayData, 37
 DisplayPeriodicData_task, 38
 exitDebugMenu, 39
 getIftPtr, 40
 getInfoStringPtr, 40
 getMenuStringPtr, 40
 info_string_ptr, 44
 isInfoStringDisplayed, 45
 MainMenuManagement, 41
 menu_string_ptr, 45
 sensorMgt_ptr, 45
 setInfoStringPtr, 42
 systemReset, 42
 WatchdogMenuManagement, 43
DebugManagement.cpp, 204
 p_global_ASW_DebugManagement, 205
 str_debug_info_message_wdg_disabled, 205
 str_debug_info_message_wdg_enabled, 206

str_debug_info_message_wdg_tmo_updated, 206
 str_debug_info_message_wdg_tmo_value, 206
 str_debug_info_message_wrong_menu_selection, 206
 str_debug_main_menu, 206
 str_debug_wdg_menu, 207
 str_debug_wdg_timeout_update_selection, 207
 DebugManagement.h, 208
 debug_mgt_main_menu_state_t, 209
 debug_mgt_wdg_state_t, 209
 p_global_ASW_DebugManagement, 210
 PERIOD_MS_TASK_DISPLAY_CPU_LOAD, 209
 PERIOD_MS_TASK_DISPLAY_DEBUG_DATA, 209
 DebugModeManagement
 DebugManagement, 36
 dht22, 46
 dht22, 47
 dht22_port, 52
 dio_ptr, 52
 getHumidity, 47
 getTemperature, 49
 initializeCommunication, 50
 mem_humidity, 52
 mem_temperature, 52
 mem_validity, 52
 pit_last_read, 52
 read, 51
 dht22.cpp, 210
 MAX_WAIT_TIME_US, 211
 p_global_BSW_dht22, 211
 dht22.h, 212
 p_global_BSW_dht22, 212
 dht22_port
 dht22, 52
 dio, 53
 dio, 54
 dio_changePortPinCnf, 54
 dio_getPort, 55
 dio_getPort_fast, 56
 dio_invertPort, 56
 dio_memorizePINaddress, 57
 dio_setPort, 58
 getDDRxAddress, 59
 getPINxAddress, 60
 getPORTxAddress, 60
 PINx_addr_mem, 61
 PINx_idx_mem, 62
 ports_init, 61
 dio.cpp, 213
 p_global_BSW_dio, 213
 dio.h, 214
 DECODE_PIN, 215
 DECODE_PORT, 215
 ENCODE_PORT, 215
 p_global_BSW_dio, 216
 PORT_CNF_IN, 215
 PORT_CNF_OUT, 216
 dio_changePortPinCnf
 dio, 54
 dio_getPort
 dio, 55
 dio_getPort_fast
 dio, 56
 dio_invertPort
 dio, 56
 dio_memorizePINaddress
 dio, 57
 dio_port_cnf.h, 216
 PORT_A, 217
 PORT_B, 217
 PORT_C, 217
 PORT_D, 218
 PORTB_CNF_DDRB, 218
 PORTB_CNF_PORTB, 218
 dio_ptr
 dht22, 52
 dio_reg_atm2560.h, 219
 DDRA_PTR, 219
 DDRB_PTR, 220
 DDRC_PTR, 220
 DDRD_PTR, 220
 PINA_PTR, 220
 PINB_PTR, 220
 PINC_PTR, 221
 PIND_PTR, 221
 PORTA_PTR, 221
 PORTB_PTR, 221
 PORTC_PTR, 221
 PORTD_PTR, 222
 dio_setPort
 dio, 58
 disable
 Watchdog, 180
 display_data
 DisplayInterface, 74
 display_en
 T_LCD_conf_struct, 157
 display_str
 T_display_data, 154
 DisplayData
 DebugManagement, 37
 DisplayFullLine
 DisplayInterface, 67, 68
 DisplayInterface, 62
 ClearFullScreen, 64
 ClearLine, 65
 ClearStringInDataStruct, 66
 display_data, 74
 DisplayFullLine, 67, 68
 DisplayInterface, 64
 dummy, 74
 FindFirstCharAddr, 68
 getDisplayDataPtr, 69
 IsLineEmpty, 70
 isShiftInProgress, 74

p_lcd, 74
RefreshLine, 70
setLineAlignment, 71
setLineAlignmentAndRefresh, 72
shiftLine_task, 72
updateLineAndRefresh, 73
DisplayInterface.cpp, 222
p_global_ASW_DisplayInterface, 223
DisplayInterface.h, 223
DISPLAY_LINE_SHIFT_PERIOD_MS, 224
DISPLAY_LINE_SHIFT_TEMPO_TIME, 224
p_global_ASW_DisplayInterface, 225
T_DisplayInterface_LineAlignment, 224
T_DisplayInterface_LineDisplayMode, 225
DisplayManagement, 75
DisplayManagement, 76
DisplaySensorData_Task, 77
GetIftPointer, 77
GetSensorMgtPtr, 78
p_SensorMgt, 79
p_display_ift, 79
RemoveWelcomeMessage_Task, 78
DisplayManagement.cpp, 226
noSensorsDisplayString, 226
p_global_ASW_DisplayManagement, 226
welcomeMessageString, 227
DisplayManagement.h, 227
DISPLAY_MGT_FIRST_LINE_SENSORS, 228
DISPLAY_MGT_I2C_BITRATE, 228
DISPLAY_MGT_LCD_I2C_ADDR, 228
DISPLAY_MGT_PERIOD_TASK_SENSOR, 228
DISPLAY_MGT_PERIOD_WELCOME_MSG_R←
EMOVAL, 229
LCD_init_cnf, 229
p_global_ASW_DisplayManagement, 229
DisplayPeriodicData_task
 DebugManagement, 38
DisplaySensorData_Task
 DisplayManagement, 77
dummy
 DisplayInterface, 74

EN_PIN
 LCD.h, 243
ENCODE_PORT
 dio.h, 215
enable
 Watchdog, 180
entryModeDir
 T_LCD_conf_struct, 158
entryModeShift
 T_LCD_conf_struct, 158
exitDebugMenu
 DebugManagement, 39

FindElement
 LinkedList, 111
FindFirstCharAddr
 DisplayInterface, 68

firstElement
 LinkedList, 115
fontType_cnf
 T_LCD_conf_struct, 158

getAverageCPUload
 CpuLoad, 22
getCurrentElement
 LinkedList, 112
getCurrentCPUload
 CpuLoad, 22
GetDDRAMAddress
 LCD, 101
getDDRxAddress
 dio, 59
getDisplayDataPtr
 DisplayInterface, 69
getFullStringFormattedValue
 SensorManagement, 134
getHumidity
 dht22, 47
GetIftPointer
 DisplayManagement, 77
getIftPtr
 DebugManagement, 40
getInfoStringPtr
 DebugManagement, 40
GetLineNumberCnf
 LCD, 101
getMaxCPUload
 CpuLoad, 23
getMenuStringPtr
 DebugManagement, 40
getMonitoringTaskPeriod
 Bmp180, 14
getPINxAddress
 dio, 60
getPORTxAddress
 dio, 60
getPitNumber
 scheduler, 118
getRawDataPtr
 Sensor, 127
getSensorCount
 SensorManagement, 135
GetSensorMgtPtr
 DisplayManagement, 78
getSensorObjectPtr
 SensorManagement, 135
getSize
 String, 146
getStatus
 Bmp180, 14
getString
 String, 147
getTMOValue
 Watchdog, 181
getTaskCount
 scheduler, 119

getTaskPeriod
 Sensor, 127

getTemperature
 dht22, 49

getTemperatureValue
 Bmp180, 14

getTimer1Value
 timer, 170

getValidity
 Sensor, 128

getValue
 Sensor, 128

getValueDecimal
 Sensor, 128

getValueInteger
 Sensor, 129

HumSensor, 80
 HumSensor, 81
 readHumSensor_task, 82
 updateTaskPeriod, 83

HumSensor.cpp, 230
 DHT22_PORT, 230

HumSensor.h, 231

I2C.cpp, 231
 p_global_BSW_i2c, 232

I2C.h, 232
 DATA_ACK, 233
 p_global_BSW_i2c, 234
 REPEATED_START, 233
 SLAR_ACK, 233
 SLAW_ACK, 234
 START, 234

I2C, 84
 bitrate, 89
 I2C, 85
 initializeBus, 85
 read, 86
 setBitRate, 87
 write, 87
 writeByte, 88

i2c_addr
 T_LCD_conf_struct, 158

i2c_bitrate
 T_LCD_conf_struct, 158

i2c_drv_ptr
 Bmp180, 19
 LCD, 107

ISR
 int.cpp, 235–237

info_string_ptr
 DebugManagement, 44

initializeBus
 I2C, 85

initializeCommunication
 dht22, 50

InitializeScreen
 LCD, 101

int.cpp, 235
 ISR, 235–237

isActive
 Watchdog, 185

isDebugActivated
 T_ASW_init_cnf, 148

isDebugModeActivated
 main.cpp, 256
 main.h, 257

isDisplayActivated
 T_ASW_init_cnf, 149

isEmpty
 T_display_data, 154

isEnabled
 Watchdog, 181

isInfoStringDisplayed
 DebugManagement, 45

isLEDActivated
 T_ASW_init_cnf, 149

IsLLEmpty
 LinkedList, 112

IsLineEmpty
 DisplayInterface, 70

isSensorMgtActivated
 T_ASW_init_cnf, 149

isShiftInProgress
 DisplayInterface, 74

keepAliveLed, 89
 blinkLed_task, 90
 keepAliveLed, 90

keepAliveLed.cpp, 238
 p_global_ASW_keepAliveLed, 238

keepAliveLed.h, 239
 LED_PORT, 239
 p_global_ASW_keepAliveLed, 240
 PERIOD_MS_TASK_LED, 240

LCD.cpp, 240
 p_global_BSW_lcd, 241

LCD.h, 241
 BACKLIGHT_PIN, 243
 EN_PIN, 243
 LCD_CNF_BACKLIGHT_OFF, 243
 LCD_CNF_BACKLIGHT_ON, 243
 LCD_CNF_CURSOR_BLINK_OFF, 244
 LCD_CNF_CURSOR_BLINK_ON, 244
 LCD_CNF_CURSOR_OFF, 244
 LCD_CNF_CURSOR_ON, 244
 LCD_CNF_DISPLAY_OFF, 244
 LCD_CNF_DISPLAY_ON, 245
 LCD_CNF_ENTRY_MODE_DIRECTION_LEFT,
 245
 LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT,
 245
 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_↔
 OFF, 245
 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_↔
 ON, 245

LCD_CNF_FONT_5_11, 246
LCD_CNF_FONT_5_8, 246
LCD_CNF_ONE_LINE, 246
LCD_CNF_SHIFT_ID, 246
LCD_CNF_SHIFT_SH, 246
LCD_CNF_TWO_LINE, 247
LCD_DISPLAY_CTRL_FIELD_B, 247
LCD_DISPLAY_CTRL_FIELD_C, 247
LCD_DISPLAY_CTRL_FIELD_D, 247
LCD_FCT_SET_FIELD_DL, 247
LCD_FCT_SET_FIELD_F, 248
LCD_FCT_SET_FIELD_N, 248
LCD_INST_CLR_DISPLAY_BIT, 248
LCD_INST_DISPLAY_CTRL, 248
LCD_INST_ENTRY_MODE_SET, 248
LCD_INST_FUNCTION_SET, 249
LCD_INST_SET_DDRAM_ADDR, 249
LCD_RAM_1_LINE_MAX, 249
LCD_RAM_1_LINE_MIN, 249
LCD_RAM_2_LINES_MAX_1, 249
LCD_RAM_2_LINES_MAX_2, 250
LCD_RAM_2_LINES_MIN_1, 250
LCD_RAM_2_LINES_MIN_2, 250
LCD_SIZE_NB_CHAR_PER_LINE, 250
LCD_SIZE_NB_LINES, 250
LCD_WAIT_CLR_RETURN, 251
LCD_WAIT_OTHER_MODES, 251
p_global_BSW_lcd, 252
RS_PIN, 251
RW_PIN, 251
T_LCD_command, 251
T_LCD_config_mode, 252
T_LCD_ram_area, 252
LCD_CNF_BACKLIGHT_OFF
 LCD.h, 243
LCD_CNF_BACKLIGHT_ON
 LCD.h, 243
LCD_CNF_CURSOR_BLINK_OFF
 LCD.h, 244
LCD_CNF_CURSOR_BLINK_ON
 LCD.h, 244
LCD_CNF_CURSOR_OFF
 LCD.h, 244
LCD_CNF_CURSOR_ON
 LCD.h, 244
LCD_CNF_DISPLAY_OFF
 LCD.h, 244
LCD_CNF_DISPLAY_ON
 LCD.h, 245
LCD_CNF_ENTRY_MODE_DIRECTION_LEFT
 LCD.h, 245
LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT
 LCD.h, 245
LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF
 LCD.h, 245
LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON
 LCD.h, 245
LCD_CNF_FONT_5_11
 LCD.h, 246
LCD_CNF_FONT_5_8
 LCD.h, 246
LCD_CNF_ONE_LINE
 LCD.h, 246
LCD_CNF_SHIFT_ID
 LCD.h, 246
LCD_CNF_SHIFT_SH
 LCD.h, 246
LCD_CNF_TWO_LINE
 LCD.h, 247
LCD_DISPLAY_CTRL_FIELD_B
 LCD.h, 247
LCD_DISPLAY_CTRL_FIELD_C
 LCD.h, 247
LCD_DISPLAY_CTRL_FIELD_D
 LCD.h, 247
LCD_FCT_SET_FIELD_DL
 LCD.h, 247
LCD_FCT_SET_FIELD_F
 LCD.h, 248
LCD_FCT_SET_FIELD_N
 LCD.h, 248
LCD_INST_CLR_DISPLAY_BIT
 LCD.h, 248
LCD_INST_DISPLAY_CTRL
 LCD.h, 248
LCD_INST_ENTRY_MODE_SET
 LCD.h, 248
LCD_INST_FUNCTION_SET
 LCD.h, 249
LCD_INST_SET_DDRAM_ADDR
 LCD.h, 249
LCD_RAM_1_LINE_MAX
 LCD.h, 249
LCD_RAM_1_LINE_MIN
 LCD.h, 249
LCD_RAM_2_LINES_MAX_1
 LCD.h, 249
LCD_RAM_2_LINES_MAX_2
 LCD.h, 250
LCD_RAM_2_LINES_MIN_1
 LCD.h, 250
LCD_RAM_2_LINES_MIN_2
 LCD.h, 250
LCD_SIZE_NB_CHAR_PER_LINE
 LCD.h, 250
LCD_SIZE_NB_LINES
 LCD.h, 250
LCD_WAIT_CLR_RETURN
 LCD.h, 251
LCD_WAIT_OTHER_MODES
 LCD.h, 251
LCD_init_cnf
 DisplayManagement.h, 229
LCD, 91
 backlight_enable, 106
 cnfCursorBlink, 106

cnfCursorOnOff, 106
 cnfDisplayOnOff, 106
 cnfEntryModeDir, 106
 cnfEntryModeShift, 106
 cnfFontType, 107
 cnfI2C_addr, 107
 cnfLineNumber, 107
 command, 94
 ConfigureBacklight, 95
 ConfigureCursorBlink, 95
 ConfigureCursorOnOff, 96
 ConfigureDisplayOnOff, 97
 ConfigureEntryModeDir, 97
 ConfigureEntryModeShift, 98
 ConfigureFontType, 99
 ConfigureI2CAddr, 99
 ConfigureLineNumber, 100
 ddram_addr, 107
 GetDDRAMAddress, 101
 GetLineNumberCnf, 101
 i2c_drv_ptr, 107
 InitializeScreen, 101
 LCD, 93
 SetDDRAMAddress, 102
 write, 103
 write4bits, 104
 WriteInRam, 105
LED_PORT
 keepAliveLed.h, 239
LLElementCompare
 scheduler, 120
last_sum_value
 CpuLoad, 24
launchPeriodicTasks
 scheduler, 119
lineNumber_cnf
 T_LCD_conf_struct, 159
LinkedList, 108
 ~LinkedList, 110
 AttachNewElement, 111
 curElement_ptr, 115
 FindElement, 111
 firstElement, 115
 getCurrentElement, 112
 IsLLEmpty, 112
 LinkedList, 110
 MoveToNextElement, 113
 RemoveElement, 113
 ResetElementPtr, 114
 T_LL_element, 109
LinkedList.cpp, 253
LinkedList.h, 253
 CompareFctPtr_t, 254
LinkedList::T_LL_element, 159
 data_ptr, 160
 nextElement, 160
MAX_WAIT_TIME_US
 dht22.cpp, 211

main
 main.cpp, 255
main.cpp, 254
 ASW_init_cnf, 256
 DEBUG_ACTIVE_PORT, 255
 isDebugModeActivated, 256
 main, 255
main.h, 257
 ASW_init_cnf, 257
 isDebugModeActivated, 257
main_state
 debug_mgt_state_struct_t, 25
MainMenuManagement
 DebugManagement, 41
max_load
 CpuLoad, 24
MB
 Bmp180::T_BMP180_calib_data, 151
MC
 Bmp180::T_BMP180_calib_data, 151
MD
 Bmp180::T_BMP180_calib_data, 152
mem_humidity
 dht22, 52
mem_temperature
 dht22, 52
mem_validity
 dht22, 52
menu_string_ptr
 DebugManagement, 45
mode
 T_display_data, 154
MoveToNextElement
 LinkedList, 113
NB_OF_SAMPLES
 CpuLoad.h, 201
nb_sensors
 SensorManagement, 137
nextElement
 LinkedList::T_LL_element, 160
nextLine
 DebugInterface, 28
noSensorsDisplayString
 DisplayManagement.cpp, 226

operator delete
 operators.cpp, 259
 operators.h, 260
 operator new
 operators.cpp, 259
 operators.h, 261
operators.cpp, 258
 operator delete, 259
 operator new, 259
operators.h, 259
 operator delete, 260
 operator new, 261

p_SensorMgt
 DisplayManagement, 79

p_display_ift
 DisplayManagement, 79

p_global_ASW_DebugInterface
 DebugInterface.cpp, 202
 DebugInterface.h, 204

p_global_ASW_DebugManagement
 DebugManagement.cpp, 205
 DebugManagement.h, 210

p_global_ASW_DisplayInterface
 DisplayInterface.cpp, 223
 DisplayInterface.h, 225

p_global_ASW_DisplayManagement
 DisplayManagement.cpp, 226
 DisplayManagement.h, 229

p_global_ASW_SensorManagement
 SensorManagement.cpp, 270
 SensorManagement.h, 271

p_global_ASW_keepAliveLed
 keepAliveLed.cpp, 238
 keepAliveLed.h, 240

p_global_BSW_bmp180
 Bmp180.cpp, 191
 Bmp180.h, 196

p_global_BSW_cpupload
 CpuLoad.cpp, 200
 CpuLoad.h, 201

p_global_BSW_dht22
 dht22.cpp, 211
 dht22.h, 212

p_global_BSW_dio
 dio.cpp, 213
 dio.h, 216

p_global_BSW_i2c
 I2C.cpp, 232
 I2C.h, 234

p_global_BSW_lcd
 LCD.cpp, 241
 LCD.h, 252

p_global_BSW_timer
 timer.cpp, 276
 timer.h, 277

p_global_BSW_usart
 usart.cpp, 278
 usart.h, 279

p_global_BSW_wdg
 Watchdog.cpp, 280
 Watchdog.h, 283

p_global_scheduler
 scheduler.cpp, 262
 scheduler.h, 264

p_lcd
 DisplayInterface, 74

PERIOD_MS_TASK_DISPLAY_CPU_LOAD
 DebugManagement.h, 209

PERIOD_MS_TASK_DISPLAY_DEBUG_DATA
 DebugManagement.h, 209

PERIOD_MS_TASK_LED
 keepAliveLed.h, 240

PINA_PTR
 dio_reg_atm2560.h, 220

PINB_PTR
 dio_reg_atm2560.h, 220

PINC_PTR
 dio_reg_atm2560.h, 221

PIND_PTR
 dio_reg_atm2560.h, 221

PINx_addr_mem
 dio, 61

PINx_idx_mem
 dio, 62

PORT_CNF_IN
 dio.h, 215

PORT_CNF_OUT
 dio.h, 216

PORT_A
 dio_port_cnf.h, 217

PORT_B
 dio_port_cnf.h, 217

PORT_C
 dio_port_cnf.h, 217

PORT_D
 dio_port_cnf.h, 218

PORTA_PTR
 dio_reg_atm2560.h, 221

PORTB_CNF_DDRB
 dio_port_cnf.h, 218

PORTB_CNF_PORTB
 dio_port_cnf.h, 218

PORTB_PTR
 dio_reg_atm2560.h, 221

PORTC_PTR
 dio_reg_atm2560.h, 221

PORTD_PTR
 dio_reg_atm2560.h, 222

PRESCALER_PERIODIC_TIMER
 scheduler.h, 263

period
 scheduler::Task_t, 162
 T_SensorManagement_Sensor_Config, 161

pit_last_read
 dht22, 52

pit_number
 scheduler, 124

ports_init
 dio, 61

prescaler1
 timer, 172

prescaler3
 timer, 172

pressure_value
 Bmp180, 19

REPEATED_START
 I2C.h, 233

RS_PIN

LCD.h, 251
 RW_PIN
 LCD.h, 251
 raw_data
 Sensor, 132
 read
 DebugInterface, 29
 dht22, 51
 I2C, 86
 readCalibData
 Bmp180, 15
 readChipID
 Bmp180, 16
 readHumSensor_task
 HumSensor, 82
 readSensor_task
 Sensor, 129
 readTempSensor_task
 TempSensor, 165
 ready
 Bmp180::T_BMP180_measurement_data, 152
 RefreshLine
 DisplayInterface, 70
 RemoveElement
 LinkedList, 113
 removePeriodicTask
 scheduler, 121
 RemoveWelcomeMessage_Task
 DisplayManagement, 78
 reset
 Watchdog, 182
 ResetElementPtr
 LinkedList, 114
 SENSOR_MGT_CNF_DEFAULT_PERIOD
 sensor_configuration.cpp, 267
 SENSOR_MGT_CNF_DEFAULT_TMO
 sensor_configuration.cpp, 267
 SLAR_ACK
 I2C.h, 233
 SLAW_ACK
 I2C.h, 234
 START
 I2C.h, 234
 SW_PERIOD_MS
 scheduler.h, 263
 sample_cnt
 CpuLoad, 24
 sample_idx
 CpuLoad, 24
 sample_mem
 CpuLoad, 25
 scheduler, 115
 addPeriodicTask, 117
 getPitNumber, 118
 getTaskCount, 119
 LLElementCompare, 120
 launchPeriodicTasks, 119
 pit_number, 124
 removePeriodicTask, 121
 scheduler, 117
 startScheduling, 122
 task_count, 124
 Task_t, 117
 TasksLL_ptr, 124
 updateTaskPeriod, 123
 scheduler.cpp, 261
 p_global_scheduler, 262
 scheduler.h, 262
 p_global_scheduler, 264
 PRESCALER_PERIODIC_TIMER, 263
 SW_PERIOD_MS, 263
 TIMER_CTC_VALUE, 263
 TaskPtr_t, 264
 scheduler::Task_t, 162
 period, 162
 TaskPtr, 162
 sendBool
 DebugInterface, 29
 sendChar
 DebugInterface, 30
 sendInteger
 DebugInterface, 31
 sendString
 DebugInterface, 32, 33
 Sensor, 125
 getRawDataPtr, 127
 getTaskPeriod, 127
 getValidity, 128
 getValue, 128
 getValueDecimal, 128
 getValueInteger, 129
 raw_data, 132
 readSensor_task, 129
 Sensor, 126
 setLastValidity, 129
 setValidityTMO, 130
 task_period, 132
 updateTaskPeriod, 130
 updateValidData, 131
 valid_pit, 132
 valid_value, 132
 validity, 132
 validity_last_read, 132
 validity_tmo, 133
 Sensor.cpp, 264
 TASK_PERIOD_DEFAULT, 265
 VALIDITY_TIMEOUT_MS_DEFAULT, 265
 Sensor.h, 265
 sensor_configuration.cpp, 266
 SENSOR_MGT_CNF_DEFAULT_PERIOD, 267
 SENSOR_MGT_CNF_DEFAULT_TMO, 267
 SensorManagement_Sensor_Config_list, 267
 sensor_configuration.h, 268
 SensorManagement_Sensor_Config_list, 269
 sensor_ptr_table
 SensorManagement, 138

sensor_type
 T_SensorManagement_Sensor_Config, 161

SensorManagement, 133
 getFullStringFormattedValue, 134
 getSensorCount, 135
 getSensorObjectPtr, 135
 nb_sensors, 137
 sensor_ptr_table, 138
 SensorManagement, 134
 updateTaskPeriod, 137

SensorManagement.cpp, 269
 p_global_ASW_SensorManagement, 270

SensorManagement.h, 270
 p_global_ASW_SensorManagement, 271
 T_SensorManagement_Sensor_Type, 271

SensorManagement_Sensor_Config_list
 sensor_configuration.cpp, 267
 sensor_configuration.h, 269

sensorMgt_ptr
 DebugManagement, 45

setBaudRate
 uart, 173

setBitRate
 I2C, 87

SetDDRAMAddress
 LCD, 102

setInfoStringPtr
 DebugManagement, 42

setLastValidity
 Sensor, 129

setLineAlignment
 DisplayInterface, 71

setLineAlignmentAndRefresh
 DisplayInterface, 72

setValidityTMO
 Sensor, 130

shift_data
 T_display_data, 154

shiftLine_task
 DisplayInterface, 72

size
 String, 147

startNewTemperatureConversion
 Bmp180, 17

startScheduling
 scheduler, 122

startTimer1
 timer, 170

startTimer3
 timer, 170

status
 Bmp180, 19

stopTimer1
 timer, 171

stopTimer3
 timer, 171

str_cur_ptr
 T_Display_shift_data, 156

str_debug_info_message_wdg_disabled
 DebugManagement.cpp, 205

str_debug_info_message_wdg_enabled
 DebugManagement.cpp, 206

str_debug_info_message_wdg_tmo_updated
 DebugManagement.cpp, 206

str_debug_info_message_wdg_tmo_value
 DebugManagement.cpp, 206

str_debug_info_message_wrong_menu_selection
 DebugManagement.cpp, 206

str_debug_main_menu
 DebugManagement.cpp, 206

str_debug_wdg_menu
 DebugManagement.cpp, 207

str_debug_wdg_timeout_update_selection
 DebugManagement.cpp, 207

str_ptr
 T_Display_shift_data, 156

String, 138
 ~String, 140
 appendBool, 141
 appendChar, 141
 appendInteger, 142
 appendSpace, 143
 appendString, 144
 Clear, 145
 ComputeStringSize, 145
 getSize, 146
 getString, 147
 size, 147
 String, 139, 140
 string, 148

string
 String, 148

String.cpp, 272

String.h, 272

SwitchWdg
 Watchdog, 182

SystemReset
 Watchdog, 183

systemReset
 DebugManagement, 42

T_ASW_init_cnf, 148
 isDebugActivated, 148
 isDisplayActivated, 149
 isLEDActivated, 149
 isSensorMgtActivated, 149

T_BMP180_status
 Bmp180.h, 195

T_Display_shift_data, 155
 str_cur_ptr, 156
 str_ptr, 156
 temporization, 156

T_DisplayInterface_LineAlignment
 DisplayInterface.h, 224

T_DisplayInterface_LineDisplayMode
 DisplayInterface.h, 225

T_LCD_command

LCD.h, 251
T_LCD_conf_struct, 156
 backlight_en, 157
 cursor_en, 157
 cursorBlink_en, 157
 display_en, 157
 entryModeDir, 158
 entryModeShift, 158
 fontType_cnf, 158
 i2c_addr, 158
 i2c_bitrate, 158
 lineNumber_cnf, 159
T_LCD_config_mode
 LCD.h, 252
T_LCD_ram_area
 LCD.h, 252
T_LL_element
 LinkedList, 109
T_SensorManagement_Sensor_Config, 160
 data_name_str, 161
 period, 161
 sensor_type, 161
 unit_str, 161
 validity_tmo, 161
T_SensorManagement_Sensor_Type
 SensorManagement.h, 271
T_display_data, 153
 alignment, 154
 display_str, 154
 isEmpty, 154
 mode, 154
 shift_data, 154
TASK_PERIOD_DEFAULT
 Sensor.cpp, 265
TEMP_SENSOR_VALUE_ACCEPTABLE_DIFFERENCE
 TempSensor.cpp, 274
TIMER_CTC_VALUE
 scheduler.h, 263
task_count
 scheduler, 124
task_period
 Bmp180, 19
 Sensor, 132
Task_t
 scheduler, 117
TaskPtr
 scheduler::Task_t, 162
TaskPtr_t
 scheduler.h, 264
TasksLL_ptr
 scheduler, 124
TempSensor, 163
 readTempSensor_task, 165
 TempSensor, 164
 updateTaskPeriod, 166
TempSensor.cpp, 273
 DHT22_PORT, 274
TEMP_SENSOR_VALUE_ACCEPTABLE_DIFFERENCE
 ERENCE, 274
TempSensor.h, 274
temperature_value
 Bmp180, 20
TemperatureMonitoring
 Bmp180, 18
temporization
 T_Display_shift_data, 156
timeoutUpdate
 Watchdog, 184
timer, 167
 configureTimer1, 168
 configureTimer3, 169
 getTimer1Value, 170
 prescaler1, 172
 prescaler3, 172
 startTimer1, 170
 startTimer3, 170
 stopTimer1, 171
 stopTimer3, 171
 timer, 168
timer.cpp, 275
 p_global_BSW_timer, 276
timer.h, 276
 p_global_BSW_timer, 277
tmo_value
 Watchdog, 185
ts
 Bmp180::T_BMP180_measurement_data, 152
USART_BAUDRATE
 DebugInterface.h, 203
unit_str
 T_SensorManagement_Sensor_Config, 161
updateLineAndRefresh
 DisplayInterface, 73
updateTaskPeriod
 HumSensor, 83
 scheduler, 123
 Sensor, 130
 SensorManagement, 137
 TempSensor, 166
updateValidData
 Sensor, 131
uart, 172
 BaudRate, 177
 setBaudRate, 173
 usart, 173
 usart_init, 174
 usart_read, 174
 usart_sendByte, 175
 usart_sendString, 176
 usart_transmit, 176
uart.cpp, 277
 p_global_BSW_usart, 278
uart.h, 278
 p_global_BSW_usart, 279
uart_drv_ptr

DebugInterface, 34
 usart_init
 USART, 174
 usart_read
 USART, 174
 usart_sendByte
 USART, 175
 usart_sendString
 USART, 176
 usart_transmit
 USART, 176

VALIDITY_TIMEOUT_MS_DEFAULT
 Sensor.cpp, 265

valid_pit
 Sensor, 132

valid_value
 Sensor, 132

validity
 Sensor, 132

validity_last_read
 Sensor, 132

validity_tmo
 Sensor, 133
 T_SensorManagement_Sensor_Config, 161

value
 Bmp180::T_BMP180_measurement_data, 153

WDG_TIMEOUT_DEFAULT_MS
 Watchdog.cpp, 280

WDG_TMO_120MS
 Watchdog.h, 281

WDG_TMO_15MS
 Watchdog.h, 281

WDG_TMO_1S
 Watchdog.h, 282

WDG_TMO_250MS
 Watchdog.h, 282

WDG_TMO_2S
 Watchdog.h, 282

WDG_TMO_30MS
 Watchdog.h, 282

WDG_TMO_4S
 Watchdog.h, 282

WDG_TMO_500MS
 Watchdog.h, 283

WDG_TMO_60MS
 Watchdog.h, 283

WDG_TMO_8S
 Watchdog.h, 283

Watchdog, 177
 disable, 180
 enable, 180
 getTMOValue, 181
 isActive, 185
 isEnabled, 181
 reset, 182
 SwitchWdg, 182
 SystemReset, 183

 timeoutUpdate, 184
 tmo_value, 185
 Watchdog, 178, 179

Watchdog.cpp, 279
 p_global_BSW_wdg, 280
 WDG_TIMEOUT_DEFAULT_MS, 280

Watchdog.h, 280
 p_global_BSW_wdg, 283
 WDG_TMO_120MS, 281
 WDG_TMO_15MS, 281
 WDG_TMO_1S, 282
 WDG_TMO_250MS, 282
 WDG_TMO_2S, 282
 WDG_TMO_30MS, 282
 WDG_TMO_4S, 282
 WDG_TMO_500MS, 283
 WDG_TMO_60MS, 283
 WDG_TMO_8S, 283

WatchdogMenuManagement
 DebugManagement, 43

wdg_state
 debug_mgt_state_struct_t, 26

welcomeMessageString
 DisplayManagement.cpp, 227

write
 I2C, 87
 LCD, 103

write4bits
 LCD, 104

writeByte
 I2C, 88

WriteInRam
 LCD, 105