

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	Member Enumeration Documentation	10
4.1.2.1	T_BMP180_conversion_type	10
4.1.2.2	T_BMP180_status	11
4.1.3	Constructor & Destructor Documentation	11
4.1.3.1	Bmp180()	11
4.1.4	Member Function Documentation	12
4.1.4.1	CalculateTemperature()	12
4.1.4.2	conversionTimerInterrupt()	13
4.1.4.3	getTemperatureValue()	13
4.1.4.4	readCalibData()	14
4.1.4.5	readChipID()	15
4.1.4.6	startNewTemperatureConversion()	16
4.1.5	Member Data Documentation	17

4.1.5.1	calibration_data	17
4.1.5.2	chip_id	17
4.1.5.3	conversionInProgress	17
4.1.5.4	i2c_drv_ptr	17
4.1.5.5	pressure_value	18
4.1.5.6	status	18
4.1.5.7	temperature_value	18
4.2	CpuLoad Class Reference	18
4.2.1	Detailed Description	19
4.2.2	Constructor & Destructor Documentation	19
4.2.2.1	CpuLoad()	19
4.2.3	Member Function Documentation	19
4.2.3.1	ComputeCPULoad()	20
4.2.3.2	getAverageCPULoad()	20
4.2.3.3	getCurrentCPULoad()	21
4.2.3.4	getMaxCPULoad()	21
4.2.4	Member Data Documentation	21
4.2.4.1	avg_load	22
4.2.4.2	current_load	22
4.2.4.3	last_sum_value	22
4.2.4.4	max_load	22
4.2.4.5	sample_cnt	22
4.2.4.6	sample_idx	23
4.2.4.7	sample_mem	23
4.3	debug_mgt_state_struct_t Struct Reference	23
4.3.1	Detailed Description	23
4.3.2	Member Data Documentation	23
4.3.2.1	main_state	24
4.3.2.2	wdg_state	24
4.4	DebugInterface Class Reference	24

4.4.1	Detailed Description	25
4.4.2	Constructor & Destructor Documentation	25
4.4.2.1	DebugInterface()	25
4.4.3	Member Function Documentation	26
4.4.3.1	ClearScreen()	26
4.4.3.2	nextLine()	26
4.4.3.3	read()	27
4.4.3.4	sendBool()	28
4.4.3.5	sendChar()	28
4.4.3.6	sendInteger()	29
4.4.3.7	sendString() [1/2]	30
4.4.3.8	sendString() [2/2]	31
4.4.4	Member Data Documentation	31
4.4.4.1	usart_drv_ptr	32
4.5	DebugManagement Class Reference	32
4.5.1	Detailed Description	33
4.5.2	Constructor & Destructor Documentation	33
4.5.2.1	DebugManagement()	34
4.5.3	Member Function Documentation	34
4.5.3.1	DebugModeManagement()	34
4.5.3.2	DisplayData()	35
4.5.3.3	DisplayPeriodicData_task()	36
4.5.3.4	exitDebugMenu()	37
4.5.3.5	getIftPtr()	38
4.5.3.6	getInfoStringPtr()	38
4.5.3.7	getMenuStringPtr()	39
4.5.3.8	MainMenuManagement()	39
4.5.3.9	setInfoStringPtr()	40
4.5.3.10	systemReset()	40
4.5.3.11	WatchdogMenuManagement()	41

4.5.4	Member Data Documentation	42
4.5.4.1	debug_ift_ptr	42
4.5.4.2	debug_state	42
4.5.4.3	info_string_ptr	43
4.5.4.4	isInfoStringDisplayed	43
4.5.4.5	menu_string_ptr	43
4.5.4.6	sensorMgt_ptr	43
4.6	dht22 Class Reference	44
4.6.1	Detailed Description	45
4.6.2	Constructor & Destructor Documentation	45
4.6.2.1	dht22()	45
4.6.3	Member Function Documentation	45
4.6.3.1	getHumidity()	46
4.6.3.2	getTemperature()	47
4.6.3.3	initializeCommunication()	48
4.6.3.4	read()	49
4.6.4	Member Data Documentation	50
4.6.4.1	dht22_port	50
4.6.4.2	dio_ptr	50
4.6.4.3	mem_humidity	50
4.6.4.4	mem_temperature	50
4.6.4.5	mem_validity	50
4.6.4.6	pit_last_read	51
4.7	dio Class Reference	51
4.7.1	Detailed Description	52
4.7.2	Constructor & Destructor Documentation	52
4.7.2.1	dio()	52
4.7.3	Member Function Documentation	52
4.7.3.1	dio_changePortPinCnf()	52
4.7.3.2	dio_getPort()	53

4.7.3.3	dio_getPort_fast()	54
4.7.3.4	dio_invertPort()	55
4.7.3.5	dio_memorizePINaddress()	55
4.7.3.6	dio_setPort()	56
4.7.3.7	getDDRxAddress()	57
4.7.3.8	getPINxAddress()	58
4.7.3.9	getPORTxAddress()	58
4.7.3.10	ports_init()	59
4.7.4	Member Data Documentation	59
4.7.4.1	PINx_addr_mem	60
4.7.4.2	PINx_idx_mem	60
4.8	DisplayInterface Class Reference	60
4.8.1	Detailed Description	62
4.8.2	Constructor & Destructor Documentation	62
4.8.2.1	DisplayInterface()	62
4.8.3	Member Function Documentation	62
4.8.3.1	ClearFullScreen()	63
4.8.3.2	ClearLine()	63
4.8.3.3	ClearStringInDataStruct()	64
4.8.3.4	DisplayFullLine() [1/2]	65
4.8.3.5	DisplayFullLine() [2/2]	66
4.8.3.6	FindFirstCharAddr()	67
4.8.3.7	getDisplayDataPtr()	67
4.8.3.8	IsLineEmpty()	68
4.8.3.9	RefreshLine()	68
4.8.3.10	setLineAlignment()	69
4.8.3.11	setLineAlignmentAndRefresh()	70
4.8.3.12	shiftLine_task()	70
4.8.3.13	updateLineAndRefresh()	71
4.8.4	Member Data Documentation	72

4.8.4.1	display_data	72
4.8.4.2	dummy	72
4.8.4.3	isShiftInProgress	72
4.8.4.4	p_lcd	73
4.9	DisplayManagement Class Reference	73
4.9.1	Detailed Description	74
4.9.2	Constructor & Destructor Documentation	74
4.9.2.1	DisplayManagement()	74
4.9.3	Member Function Documentation	75
4.9.3.1	DisplaySensorData_Task()	75
4.9.3.2	GetIftPointer()	76
4.9.3.3	GetSensorMgtPtr()	76
4.9.3.4	RemoveWelcomeMessage_Task()	77
4.9.4	Member Data Documentation	77
4.9.4.1	p_display_ift	77
4.9.4.2	p_SensorMgt	78
4.10	HumSensor Class Reference	78
4.10.1	Detailed Description	79
4.10.2	Constructor & Destructor Documentation	79
4.10.2.1	HumSensor() [1/2]	79
4.10.2.2	HumSensor() [2/2]	80
4.10.3	Member Function Documentation	80
4.10.3.1	readHumSensor_task()	80
4.10.3.2	updateTaskPeriod()	81
4.11	I2C Class Reference	82
4.11.1	Detailed Description	83
4.11.2	Constructor & Destructor Documentation	83
4.11.2.1	I2C()	83
4.11.3	Member Function Documentation	83
4.11.3.1	initializeBus()	84

4.11.3.2 <code>read()</code>	84
4.11.3.3 <code>setBitRate()</code>	85
4.11.3.4 <code>write()</code>	85
4.11.3.5 <code>writeByte()</code>	86
4.11.4 Member Data Documentation	87
4.11.4.1 <code>bitrate</code>	87
4.12 keepAliveLed Class Reference	87
4.12.1 Detailed Description	88
4.12.2 Constructor & Destructor Documentation	88
4.12.2.1 <code>keepAliveLed()</code>	88
4.12.3 Member Function Documentation	88
4.12.3.1 <code>blinkLed_task()</code>	89
4.13 LCD Class Reference	89
4.13.1 Detailed Description	91
4.13.2 Constructor & Destructor Documentation	91
4.13.2.1 <code>LCD()</code>	91
4.13.3 Member Function Documentation	92
4.13.3.1 <code>command()</code>	92
4.13.3.2 <code>ConfigureBacklight()</code>	93
4.13.3.3 <code>ConfigureCursorBlink()</code>	94
4.13.3.4 <code>ConfigureCursorOnOff()</code>	94
4.13.3.5 <code>ConfigureDisplayOnOff()</code>	95
4.13.3.6 <code>ConfigureEntryModeDir()</code>	95
4.13.3.7 <code>ConfigureEntryModeShift()</code>	96
4.13.3.8 <code>ConfigureFontType()</code>	97
4.13.3.9 <code>ConfigureI2CAddr()</code>	98
4.13.3.10 <code>ConfigureLineNumber()</code>	98
4.13.3.11 <code>GetDDRAMAddress()</code>	99
4.13.3.12 <code>GetLineNumberCnf()</code>	99
4.13.3.13 <code>InitializeScreen()</code>	100

4.13.3.14 SetDDRAMAddress()	100
4.13.3.15 write()	101
4.13.3.16 write4bits()	102
4.13.3.17 WriteInRam()	103
4.13.4 Member Data Documentation	104
4.13.4.1 backlight_enable	104
4.13.4.2 cnfCursorBlink	104
4.13.4.3 cnfCursorOnOff	104
4.13.4.4 cnfDisplayOnOff	104
4.13.4.5 cnfEntryModeDir	104
4.13.4.6 cnfEntryModeShift	105
4.13.4.7 cnfFontType	105
4.13.4.8 cnfI2C_addr	105
4.13.4.9 cnfLineNumber	105
4.13.4.10 ddrum_addr	105
4.13.4.11 i2c_drv_ptr	106
4.14 LinkedList Class Reference	106
4.14.1 Detailed Description	107
4.14.2 Member Typedef Documentation	107
4.14.2.1 T_LL_element	108
4.14.3 Constructor & Destructor Documentation	108
4.14.3.1 LinkedList()	108
4.14.3.2 ~LinkedList()	108
4.14.4 Member Function Documentation	109
4.14.4.1 AttachNewElement()	109
4.14.4.2 FindElement()	109
4.14.4.3 getCurrentElement()	110
4.14.4.4 IsLLEmpty()	111
4.14.4.5 MoveToNextElement()	111
4.14.4.6 RemoveElement()	112

4.14.4.7	ResetElementPtr()	112
4.14.5	Member Data Documentation	113
4.14.5.1	curElement_ptr	113
4.14.5.2	firstElement	113
4.15	scheduler Class Reference	113
4.15.1	Detailed Description	114
4.15.2	Member Typedef Documentation	115
4.15.2.1	Task_t	115
4.15.3	Constructor & Destructor Documentation	115
4.15.3.1	scheduler()	115
4.15.4	Member Function Documentation	115
4.15.4.1	addPeriodicTask()	115
4.15.4.2	getPitNumber()	116
4.15.4.3	getTaskCount()	117
4.15.4.4	launchPeriodicTasks()	117
4.15.4.5	LLElementCompare()	118
4.15.4.6	removePeriodicTask()	119
4.15.4.7	startScheduling()	120
4.15.4.8	updateTaskPeriod()	121
4.15.5	Member Data Documentation	122
4.15.5.1	pit_number	122
4.15.5.2	task_count	122
4.15.5.3	TasksLL_ptr	122
4.16	Sensor Class Reference	123
4.16.1	Detailed Description	124
4.16.2	Constructor & Destructor Documentation	124
4.16.2.1	Sensor() [1/2]	124
4.16.2.2	Sensor() [2/2]	124
4.16.3	Member Function Documentation	125
4.16.3.1	getRawDataPtr()	125

4.16.3.2	getTaskPeriod()	126
4.16.3.3	getValidity()	126
4.16.3.4	getValue()	126
4.16.3.5	getValueDecimal()	127
4.16.3.6	getValueInteger()	127
4.16.3.7	readSensor_task()	127
4.16.3.8	setLastValidity()	127
4.16.3.9	setValidityTMO()	128
4.16.3.10	updateTaskPeriod()	129
4.16.3.11	updateValidData()	129
4.16.4	Member Data Documentation	130
4.16.4.1	raw_data	130
4.16.4.2	task_period	130
4.16.4.3	valid_pit	130
4.16.4.4	valid_value	130
4.16.4.5	validity	130
4.16.4.6	validity_last_read	131
4.16.4.7	validity_tmo	131
4.17	SensorManagement Class Reference	131
4.17.1	Detailed Description	132
4.17.2	Constructor & Destructor Documentation	132
4.17.2.1	SensorManagement()	132
4.17.3	Member Function Documentation	132
4.17.3.1	getFullStringFormattedValue()	132
4.17.3.2	getSensorCount()	133
4.17.3.3	getSensorObjectPtr()	134
4.17.3.4	updateTaskPeriod()	135
4.17.4	Member Data Documentation	135
4.17.4.1	nb_sensors	136
4.17.4.2	sensor_ptr_table	136

4.18 String Class Reference	136
4.18.1 Detailed Description	137
4.18.2 Constructor & Destructor Documentation	137
4.18.2.1 <code>String() [1/2]</code>	137
4.18.2.2 <code>String() [2/2]</code>	138
4.18.2.3 <code>~String()</code>	138
4.18.3 Member Function Documentation	139
4.18.3.1 <code>appendBool()</code>	139
4.18.3.2 <code>appendChar()</code>	140
4.18.3.3 <code>appendInteger()</code>	140
4.18.3.4 <code>appendSpace()</code>	141
4.18.3.5 <code>appendString()</code>	142
4.18.3.6 <code>Clear()</code>	143
4.18.3.7 <code>ComputeStringSize()</code>	144
4.18.3.8 <code>getSize()</code>	144
4.18.3.9 <code>getString()</code>	145
4.18.4 Member Data Documentation	145
4.18.4.1 <code>size</code>	145
4.18.4.2 <code>string</code>	145
4.19 T_ASW_init_cnf Struct Reference	146
4.19.1 Detailed Description	146
4.19.2 Member Data Documentation	146
4.19.2.1 <code>isDebugActivated</code>	146
4.19.2.2 <code>isDisplayActivated</code>	146
4.19.2.3 <code>isLEDActivated</code>	147
4.19.2.4 <code>isSensorMgtActivated</code>	147
4.20 Bmp180::T_BMP180_calib_data Struct Reference	147
4.20.1 Detailed Description	147
4.20.2 Member Data Documentation	148
4.20.2.1 <code>AC1</code>	148

4.20.2.2	AC2	148
4.20.2.3	AC3	148
4.20.2.4	AC4	148
4.20.2.5	AC5	148
4.20.2.6	AC6	149
4.20.2.7	B1	149
4.20.2.8	B2	149
4.20.2.9	MB	149
4.20.2.10	MC	149
4.20.2.11	MD	149
4.21	Bmp180::T_BMP180_measurement_data Struct Reference	150
4.21.1	Detailed Description	150
4.21.2	Member Data Documentation	150
4.21.2.1	ready	150
4.21.2.2	ts	150
4.21.2.3	value	150
4.22	T_display_data Struct Reference	151
4.22.1	Detailed Description	151
4.22.2	Member Data Documentation	151
4.22.2.1	alignment	152
4.22.2.2	display_str	152
4.22.2.3	isEmpty	152
4.22.2.4	mode	152
4.22.2.5	shift_data	152
4.23	T_Display_shift_data Struct Reference	153
4.23.1	Detailed Description	153
4.23.2	Member Data Documentation	153
4.23.2.1	str_cur_ptr	153
4.23.2.2	str_ptr	154
4.23.2.3	temporization	154

4.24 T_LCD_conf_struct Struct Reference	154
4.24.1 Detailed Description	154
4.24.2 Member Data Documentation	155
4.24.2.1 backlight_en	155
4.24.2.2 cursor_en	155
4.24.2.3 cursorBlink_en	155
4.24.2.4 display_en	155
4.24.2.5 entryModeDir	155
4.24.2.6 entryModeShift	156
4.24.2.7 fontType_cnf	156
4.24.2.8 i2c_addr	156
4.24.2.9 i2c_bitrate	156
4.24.2.10 lineNumber_cnf	156
4.25 LinkedList::T_LL_element Struct Reference	157
4.25.1 Detailed Description	157
4.25.2 Member Data Documentation	157
4.25.2.1 data_ptr	157
4.25.2.2 nextElement	157
4.26 T_SensorManagement_Sensor_Config Struct Reference	158
4.26.1 Detailed Description	158
4.26.2 Member Data Documentation	158
4.26.2.1 data_name_str	158
4.26.2.2 period	158
4.26.2.3 sensor_type	158
4.26.2.4 unit_str	159
4.26.2.5 validity_tmo	159
4.27 scheduler::Task_t Struct Reference	159
4.27.1 Detailed Description	159
4.27.2 Member Data Documentation	159
4.27.2.1 period	159

4.27.2.2 TaskPtr	160
4.28 TempSensor Class Reference	160
4.28.1 Detailed Description	161
4.28.2 Constructor & Destructor Documentation	161
4.28.2.1 TempSensor() [1/2]	161
4.28.2.2 TempSensor() [2/2]	162
4.28.3 Member Function Documentation	162
4.28.3.1 readTempSensor_task()	162
4.28.3.2 updateTaskPeriod()	163
4.29 timer Class Reference	164
4.29.1 Detailed Description	165
4.29.2 Constructor & Destructor Documentation	165
4.29.2.1 timer()	165
4.29.3 Member Function Documentation	165
4.29.3.1 configureTimer1()	165
4.29.3.2 configureTimer3()	166
4.29.3.3 getTimer1Value()	167
4.29.3.4 startTimer1()	167
4.29.3.5 startTimer3()	168
4.29.3.6 stopTimer1()	168
4.29.3.7 stopTimer3()	168
4.29.4 Member Data Documentation	169
4.29.4.1 prescaler1	169
4.29.4.2 prescaler3	169
4.30 usart Class Reference	169
4.30.1 Detailed Description	170
4.30.2 Constructor & Destructor Documentation	170
4.30.2.1 usart()	170
4.30.3 Member Function Documentation	170
4.30.3.1 setBaudRate()	171

4.30.3.2 uart_init()	171
4.30.3.3 uart_read()	172
4.30.3.4 uart_sendByte()	172
4.30.3.5 uart_sendString()	173
4.30.3.6 uart_transmit()	174
4.30.4 Member Data Documentation	174
4.30.4.1 BaudRate	174
4.31 Watchdog Class Reference	174
4.31.1 Detailed Description	175
4.31.2 Constructor & Destructor Documentation	175
4.31.2.1 Watchdog() [1/2]	176
4.31.2.2 Watchdog() [2/2]	176
4.31.3 Member Function Documentation	177
4.31.3.1 disable()	177
4.31.3.2 enable()	177
4.31.3.3 getTMOValue()	178
4.31.3.4 isEnabled()	179
4.31.3.5 reset()	179
4.31.3.6 SwitchWdg()	180
4.31.3.7 SystemReset()	180
4.31.3.8 timeoutUpdate()	181
4.31.4 Member Data Documentation	182
4.31.4.1 isActive	182
4.31.4.2 tmo_value	182

5 File Documentation	183
5.1 asw.cpp File Reference	183
5.1.1 Detailed Description	184
5.1.2 Function Documentation	184
5.1.2.1 asw_init()	184
5.2 asw.h File Reference	185
5.2.1 Detailed Description	185
5.2.2 Function Documentation	185
5.2.2.1 asw_init()	186
5.3 Bmp180.cpp File Reference	186
5.3.1 Detailed Description	187
5.3.2 Variable Documentation	187
5.3.2.1 p_global_BSW_bmp180	187
5.4 Bmp180.h File Reference	188
5.4.1 Detailed Description	189
5.4.2 Macro Definition Documentation	189
5.4.2.1 BMP180_CHIP_ID_CALIB_EEP_START_ADDR	189
5.4.2.2 BMP180_CHIP_ID_EEP_ADDR	189
5.4.2.3 BMP180_CHIP_ID_EXPECTED	189
5.4.2.4 BMP180_CTRL_MEAS_EEP_ADDR	190
5.4.2.5 BMP180_CTRL_MEAS_START_TEMP_CONV	190
5.4.2.6 BMP180_DATA_VALIDITY_TMO	190
5.4.2.7 BMP180_I2C_ADDR	190
5.4.2.8 BMP180_I2C_BITRATE	190
5.4.2.9 BMP180_OUT_REG_LSB EEPROM_ADDR	191
5.4.2.10 BMP180_OUT_REG_MSB EEPROM_ADDR	191
5.4.2.11 BMP180_TEMP_MEAS_TIMER_CTC_VALUE	191
5.4.2.12 BMP180_TIMER_PRESCALER_VALUE	191
5.4.3 Variable Documentation	191
5.4.3.1 p_global_BSW_bmp180	191

5.5 bsw.cpp File Reference	192
5.5.1 Detailed Description	192
5.5.2 Function Documentation	192
5.5.2.1 bsw_init()	193
5.6 bsw.h File Reference	193
5.6.1 Detailed Description	194
5.6.2 Function Documentation	194
5.6.2.1 bsw_init()	194
5.7 CpuLoad.cpp File Reference	195
5.7.1 Detailed Description	195
5.7.2 Variable Documentation	195
5.7.2.1 p_global_BSW_cpupload	195
5.8 CpuLoad.h File Reference	196
5.8.1 Detailed Description	196
5.8.2 Macro Definition Documentation	196
5.8.2.1 NB_OF_SAMPLES	197
5.8.3 Variable Documentation	197
5.8.3.1 p_global_BSW_cpupload	197
5.9 DebugInterface.cpp File Reference	197
5.9.1 Detailed Description	198
5.9.2 Variable Documentation	198
5.9.2.1 p_global_ASW_DebugInterface	198
5.10 DebugInterface.h File Reference	198
5.10.1 Detailed Description	199
5.10.2 Macro Definition Documentation	199
5.10.2.1 USART_BAUDRATE	199
5.10.3 Variable Documentation	199
5.10.3.1 p_global_ASW_DebugInterface	199
5.11 DebugManagement.cpp File Reference	200
5.11.1 Detailed Description	201

5.11.2 Variable Documentation	201
5.11.2.1 p_global_ASW_DebugManagement	201
5.11.2.2 str_debug_info_message_wdg_disabled	201
5.11.2.3 str_debug_info_message_wdg_enabled	201
5.11.2.4 str_debug_info_message_wdg_tmo_updated	202
5.11.2.5 str_debug_info_message_wdg_tmo_value	202
5.11.2.6 str_debug_info_message_wrong_menu_selection	202
5.11.2.7 str_debug_main_menu	202
5.11.2.8 str_debug_wdg_menu	203
5.11.2.9 str_debug_wdg_timeout_update_selection	203
5.12 DebugManagement.h File Reference	203
5.12.1 Detailed Description	204
5.12.2 Macro Definition Documentation	204
5.12.2.1 PERIOD_MS_TASK_DISPLAY_CPU_LOAD	204
5.12.2.2 PERIOD_MS_TASK_DISPLAY_DEBUG_DATA	205
5.12.3 Enumeration Type Documentation	205
5.12.3.1 debug_mgt_main_menu_state_t	205
5.12.3.2 debug_mgt_wdg_state_t	205
5.12.4 Variable Documentation	205
5.12.4.1 p_global_ASW_DebugManagement	206
5.13 dht22.cpp File Reference	206
5.13.1 Detailed Description	206
5.13.2 Macro Definition Documentation	207
5.13.2.1 MAX_WAIT_TIME_US	207
5.13.3 Variable Documentation	207
5.13.3.1 p_global_BSW_dht22	207
5.14 dht22.h File Reference	207
5.14.1 Detailed Description	208
5.14.2 Variable Documentation	208
5.14.2.1 p_global_BSW_dht22	208

5.15 dio.cpp File Reference	208
5.15.1 Detailed Description	209
5.15.2 Variable Documentation	209
5.15.2.1 p_global_BSW_dio	209
5.16 dio.h File Reference	209
5.16.1 Detailed Description	210
5.16.2 Macro Definition Documentation	210
5.16.2.1 DECODE_PIN	211
5.16.2.2 DECODE_PORT	211
5.16.2.3 ENCODE_PORT	211
5.16.2.4 PORT_CNF_IN	211
5.16.2.5 PORT_CNF_OUT	211
5.16.3 Variable Documentation	212
5.16.3.1 p_global_BSW_dio	212
5.17 dio_port_cnf.h File Reference	212
5.17.1 Detailed Description	212
5.17.2 Macro Definition Documentation	213
5.17.2.1 PORT_A	213
5.17.2.2 PORT_B	213
5.17.2.3 PORT_C	213
5.17.2.4 PORT_D	213
5.17.2.5 PORTB_CNF_DDRB	214
5.17.2.6 PORTB_CNF_PORTB	214
5.18 dio_reg_atm2560.h File Reference	214
5.18.1 Detailed Description	215
5.18.2 Macro Definition Documentation	215
5.18.2.1 DDRA_PTR	215
5.18.2.2 DDRB_PTR	215
5.18.2.3 DDRC_PTR	216
5.18.2.4 DDRD_PTR	216

5.18.2.5 PINA_PTR	216
5.18.2.6 PINB_PTR	216
5.18.2.7 PINC_PTR	216
5.18.2.8 PIND_PTR	217
5.18.2.9 PORTA_PTR	217
5.18.2.10 PORTB_PTR	217
5.18.2.11 PORTC_PTR	217
5.18.2.12 PORTD_PTR	217
5.19 DisplayInterface.cpp File Reference	218
5.19.1 Detailed Description	218
5.19.2 Variable Documentation	218
5.19.2.1 p_global_ASW_DisplayInterface	218
5.20 DisplayInterface.h File Reference	219
5.20.1 Detailed Description	220
5.20.2 Macro Definition Documentation	220
5.20.2.1 DISPLAY_LINE_SHIFT_PERIOD_MS	220
5.20.2.2 DISPLAY_LINE_SHIFT_TEMPO_TIME	220
5.20.3 Enumeration Type Documentation	220
5.20.3.1 T_DisplayInterface_LineAlignment	220
5.20.3.2 T_DisplayInterface_LineDisplayMode	221
5.20.4 Variable Documentation	221
5.20.4.1 p_global_ASW_DisplayInterface	221
5.21 DisplayManagement.cpp File Reference	222
5.21.1 Detailed Description	222
5.21.2 Variable Documentation	222
5.21.2.1 noSensorsDisplayString	222
5.21.2.2 p_global_ASW_DisplayManagement	223
5.21.2.3 welcomeMessageString	223
5.22 DisplayManagement.h File Reference	223
5.22.1 Detailed Description	224

5.22.2 Macro Definition Documentation	224
5.22.2.1 DISPLAY_MGT_FIRST_LINE_SENSORS	224
5.22.2.2 DISPLAY_MGT_I2C_BITRATE	224
5.22.2.3 DISPLAY_MGT_LCD_I2C_ADDR	224
5.22.2.4 DISPLAY_MGT_PERIOD_TASK_SENSOR	225
5.22.2.5 DISPLAY_MGT_PERIOD_WELCOME_MSG_REMOVAL	225
5.22.3 Variable Documentation	225
5.22.3.1 LCD_init_cnf	225
5.22.3.2 p_global_ASW_DisplayManagement	225
5.23 HumSensor.cpp File Reference	226
5.23.1 Detailed Description	226
5.23.2 Macro Definition Documentation	226
5.23.2.1 DHT22_PORT	226
5.24 HumSensor.h File Reference	227
5.24.1 Detailed Description	227
5.25 I2C.cpp File Reference	227
5.25.1 Detailed Description	228
5.25.2 Variable Documentation	228
5.25.2.1 p_global_BSW_i2c	228
5.26 I2C.h File Reference	228
5.26.1 Detailed Description	229
5.26.2 Macro Definition Documentation	229
5.26.2.1 DATA_ACK	229
5.26.2.2 REPEATED_START	229
5.26.2.3 SLAR_ACK	230
5.26.2.4 SLAW_ACK	230
5.26.2.5 START	230
5.26.3 Variable Documentation	230
5.26.3.1 p_global_BSW_i2c	230
5.27 int.cpp File Reference	231

5.27.1	Detailed Description	231
5.27.2	Function Documentation	231
5.27.2.1	ISR() [1/3]	232
5.27.2.2	ISR() [2/3]	232
5.27.2.3	ISR() [3/3]	233
5.28	keepAliveLed.cpp File Reference	234
5.28.1	Detailed Description	234
5.28.2	Variable Documentation	234
5.28.2.1	p_global_ASW_keepAliveLed	234
5.29	keepAliveLed.h File Reference	235
5.29.1	Detailed Description	235
5.29.2	Macro Definition Documentation	235
5.29.2.1	LED_PORT	236
5.29.2.2	PERIOD_MS_TASK_LED	236
5.29.3	Variable Documentation	236
5.29.3.1	p_global_ASW_keepAliveLed	236
5.30	LCD.cpp File Reference	236
5.30.1	Detailed Description	237
5.30.2	Variable Documentation	237
5.30.2.1	p_global_BSW_lcd	237
5.31	LCD.h File Reference	237
5.31.1	Detailed Description	239
5.31.2	Macro Definition Documentation	239
5.31.2.1	BACKLIGHT_PIN	239
5.31.2.2	EN_PIN	239
5.31.2.3	LCD_CNF_BACKLIGHT_OFF	239
5.31.2.4	LCD_CNF_BACKLIGHT_ON	240
5.31.2.5	LCD_CNF_CURSOR_BLINK_OFF	240
5.31.2.6	LCD_CNF_CURSOR_BLINK_ON	240
5.31.2.7	LCD_CNF_CURSOR_OFF	240

5.31.2.8 LCD_CNF_CURSOR_ON	240
5.31.2.9 LCD_CNF_DISPLAY_OFF	241
5.31.2.10 LCD_CNF_DISPLAY_ON	241
5.31.2.11 LCD_CNF_ENTRY_MODE_DIRECTION_LEFT	241
5.31.2.12 LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT	241
5.31.2.13 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF	241
5.31.2.14 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON	242
5.31.2.15 LCD_CNF_FONT_5_11	242
5.31.2.16 LCD_CNF_FONT_5_8	242
5.31.2.17 LCD_CNF_ONE_LINE	242
5.31.2.18 LCD_CNF_SHIFT_ID	242
5.31.2.19 LCD_CNF_SHIFT_SH	243
5.31.2.20 LCD_CNF_TWO_LINE	243
5.31.2.21 LCD_DISPLAY_CTRL_FIELD_B	243
5.31.2.22 LCD_DISPLAY_CTRL_FIELD_C	243
5.31.2.23 LCD_DISPLAY_CTRL_FIELD_D	243
5.31.2.24 LCD_FCT_SET_FIELD_DL	244
5.31.2.25 LCD_FCT_SET_FIELD_F	244
5.31.2.26 LCD_FCT_SET_FIELD_N	244
5.31.2.27 LCD_INST_CLR_DISPLAY_BIT	244
5.31.2.28 LCD_INST_DISPLAY_CTRL	244
5.31.2.29 LCD_INST_ENTRY_MODE_SET	245
5.31.2.30 LCD_INST_FUNCTION_SET	245
5.31.2.31 LCD_INST_SET_DDRAM_ADDR	245
5.31.2.32 LCD_RAM_1_LINE_MAX	245
5.31.2.33 LCD_RAM_1_LINE_MIN	245
5.31.2.34 LCD_RAM_2_LINES_MAX_1	246
5.31.2.35 LCD_RAM_2_LINES_MAX_2	246
5.31.2.36 LCD_RAM_2_LINES_MIN_1	246
5.31.2.37 LCD_RAM_2_LINES_MIN_2	246

5.31.2.38 LCD_SIZE_NB_CHAR_PER_LINE	246
5.31.2.39 LCD_SIZE_NB_LINES	247
5.31.2.40 LCD_WAIT_CLR_RETURN	247
5.31.2.41 LCD_WAIT_OTHER_MODES	247
5.31.2.42 RS_PIN	247
5.31.2.43 RW_PIN	247
5.31.3 Enumeration Type Documentation	247
5.31.3.1 T_LCD_command	247
5.31.3.2 T_LCD_config_mode	248
5.31.3.3 T_LCD_ram_area	248
5.31.4 Variable Documentation	248
5.31.4.1 p_global_BSW_lcd	249
5.32 LinkedList.cpp File Reference	249
5.32.1 Detailed Description	249
5.33 LinkedList.h File Reference	249
5.33.1 Detailed Description	250
5.33.2 Typedef Documentation	250
5.33.2.1 CompareFctPtr_t	250
5.34 main.cpp File Reference	250
5.34.1 Detailed Description	251
5.34.2 Macro Definition Documentation	251
5.34.2.1 DEBUG_ACTIVE_PORT	251
5.34.3 Function Documentation	251
5.34.3.1 main()	252
5.34.4 Variable Documentation	252
5.34.4.1 ASW_init_cnf	252
5.34.4.2 isDebugModeActivated	252
5.35 main.h File Reference	253
5.35.1 Detailed Description	253
5.35.2 Variable Documentation	253

5.35.2.1 ASW_init_cnf	253
5.35.2.2 isDebugModeActivated	254
5.36 operators.cpp File Reference	254
5.36.1 Detailed Description	254
5.36.2 Function Documentation	255
5.36.2.1 operator delete()	255
5.36.2.2 operator new()	255
5.37 operators.h File Reference	255
5.37.1 Detailed Description	256
5.37.2 Function Documentation	256
5.37.2.1 operator delete()	256
5.37.2.2 operator new()	257
5.38 scheduler.cpp File Reference	257
5.38.1 Detailed Description	258
5.38.2 Variable Documentation	258
5.38.2.1 p_global_scheduler	258
5.39 scheduler.h File Reference	258
5.39.1 Detailed Description	259
5.39.2 Macro Definition Documentation	259
5.39.2.1 PRESCALER_PERIODIC_TIMER	259
5.39.2.2 SW_PERIOD_MS	259
5.39.2.3 TIMER_CTC_VALUE	260
5.39.3 Typedef Documentation	260
5.39.3.1 TaskPtr_t	260
5.39.4 Variable Documentation	260
5.39.4.1 p_global_scheduler	260
5.40 Sensor.cpp File Reference	260
5.40.1 Detailed Description	261
5.40.2 Macro Definition Documentation	261
5.40.2.1 TASK_PERIOD_DEFAULT	261

5.40.2.2 VALIDITY_TIMEOUT_MS_DEFAULT	261
5.41 Sensor.h File Reference	261
5.41.1 Detailed Description	262
5.42 sensor_configuration.cpp File Reference	262
5.42.1 Detailed Description	263
5.42.2 Macro Definition Documentation	263
5.42.2.1 SENSOR_MGT_CNF_DEFAULT_PERIOD	263
5.42.2.2 SENSOR_MGT_CNF_DEFAULT_TMO	263
5.42.3 Variable Documentation	263
5.42.3.1 SensorManagement_Sensor_Config_list	264
5.43 sensor_configuration.h File Reference	264
5.43.1 Detailed Description	265
5.43.2 Variable Documentation	265
5.43.2.1 SensorManagement_Sensor_Config_list	265
5.44 SensorManagement.cpp File Reference	265
5.44.1 Detailed Description	266
5.44.2 Variable Documentation	266
5.44.2.1 p_global_ASW_SensorManagement	266
5.45 SensorManagement.h File Reference	266
5.45.1 Detailed Description	267
5.45.2 Enumeration Type Documentation	267
5.45.2.1 T_SensorManagement_Sensor_Type	267
5.45.3 Variable Documentation	267
5.45.3.1 p_global_ASW_SensorManagement	267
5.46 String.cpp File Reference	268
5.46.1 Detailed Description	268
5.47 String.h File Reference	268
5.47.1 Detailed Description	269
5.48 TempSensor.cpp File Reference	269
5.48.1 Detailed Description	269

5.48.2 Macro Definition Documentation	270
5.48.2.1 DHT22_PORT	270
5.48.2.2 TEMP_SENSOR_VALUE_ACCEPTABLE_DIFFERENCE	270
5.49 TempSensor.h File Reference	270
5.49.1 Detailed Description	271
5.50 timer.cpp File Reference	271
5.50.1 Detailed Description	271
5.50.2 Variable Documentation	272
5.50.2.1 p_global_BSW_timer	272
5.51 timer.h File Reference	272
5.51.1 Detailed Description	272
5.51.2 Variable Documentation	273
5.51.2.1 p_global_BSW_timer	273
5.52 usart.cpp File Reference	273
5.52.1 Detailed Description	273
5.52.2 Variable Documentation	274
5.52.2.1 p_global_BSW_usart	274
5.53 usart.h File Reference	274
5.53.1 Detailed Description	274
5.53.2 Variable Documentation	275
5.53.2.1 p_global_BSW_usart	275
5.54 Watchdog.cpp File Reference	275
5.54.1 Detailed Description	276
5.54.2 Macro Definition Documentation	276
5.54.2.1 WDG_TIMEOUT_DEFAULT_MS	276
5.54.3 Variable Documentation	276
5.54.3.1 p_global_BSW_wdg	276
5.55 Watchdog.h File Reference	276
5.55.1 Detailed Description	277
5.55.2 Macro Definition Documentation	277
5.55.2.1 WDG_TMO_120MS	277
5.55.2.2 WDG_TMO_15MS	278
5.55.2.3 WDG_TMO_1S	278
5.55.2.4 WDG_TMO_250MS	278
5.55.2.5 WDG_TMO_2S	278
5.55.2.6 WDG_TMO_30MS	278
5.55.2.7 WDG_TMO_4S	279
5.55.2.8 WDG_TMO_500MS	279
5.55.2.9 WDG_TMO_60MS	279
5.55.2.10 WDG_TMO_8S	279
5.55.3 Variable Documentation	279
5.55.3.1 p_global_BSW_wdg	279

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Bmp180	9
CpuLoad	18
debug_mgt_state_struct_t	23
DebugInterface	24
DebugManagement	32
dht22	44
dio	51
DisplayInterface	60
DisplayManagement	73
I2C	82
keepAliveLed	87
LCD	89
LinkedList	106
scheduler	113
Sensor	123
HumSensor	78
TempSensor	160
SensorManagement	131
String	136
T_ASW_init_cnf	146
Bmp180::T_BMP180_calib_data	147
Bmp180::T_BMP180_measurement_data	150
T_display_data	151
T_Display_shift_data	153
T_LCD_conf_struct	154
LinkedList::T_LL_element	157
T_SensorManagement_Sensor_Config	158
scheduler::Task_t	159
timer	164
uart	169
Watchdog	174

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	18
debug_mgt_state_struct_t	Structure containing all debug states	23
DebugInterface	Class used for debugging on usart link	24
DebugManagement	Debug management class	32
dht22	DHT 22 driver class	44
dio	DIO class	51
DisplayInterface	Display interface services class	60
DisplayManagement	Display management class	73
HumSensor	Class for humidity sensor	78
I2C	Two-wire serial interface (I2C) class definition	82
keepAliveLed	Class for keep-alive LED blinking	87
LCD	Class for LCD S2004A display driver	89
LinkedList	Linked list class	106
scheduler	Scheduler class	113
Sensor	Generic class for sensor device	123
SensorManagement	Sensor management class	131
String	String management class	136

T_ASW_init_cnf	ASW initialization configuration structure	146
Bmp180::T_BMP180_calib_data	Structure defining the calibration data of BMP180 sensor	147
Bmp180::T_BMP180_measurement_data	Structure defining a sensor value and its status	150
T_display_data	Structure containing display data	151
T_Display_shift_data	Structure containing shift data	153
T_LCD_conf_struct	Structure defining LCD configuration	154
LinkedList::T_LL_element	Type defining a linked list element	157
T_SensorManagement_Sensor_Config	Sensor informations structure	158
scheduler::Task_t	Type defining a task structure	159
TempSensor	Class for temperature sensor	160
timer	Class defining a timer	164
uart	USART serial bus class	169
Watchdog	Watchdog management class	174

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

asw.cpp	ASW main file	183
asw.h	ASW main header file	185
Bmp180.cpp	Bmp180 class source file	186
Bmp180.h	Bmp180 class header file	188
bsw.cpp	BSW main file	192
bsw.h	BSW main header file	193
CpuLoad.cpp	Defines functions of class CpuLoad	195
CpuLoad.h	CpuLoad class header file	196
DebugInterface.cpp	This file defines classes for log and debug data transmission on USART link	197
DebugInterface.h	Header file for debug and logging functions	198
DebugManagement.cpp	Debug management class source file	200
DebugManagement.h	Debug management class header file	203
dht22.cpp	This file defines classes for DHT22 driver	206
dht22.h	DHT22 driver header file	207
dio.cpp	DIO library	208
dio.h	DIO library header file	209
dio_port_cnf.h	Digital ports configuration file	212
dio_reg_atm2560.h	Defines DIO register addresses for ATMEGA2560	214

DisplayInterface.cpp	Source code file for display services	218
DisplayInterface.h	DisplayInterface class header file	219
DisplayManagement.cpp	Display management source file	222
DisplayManagement.h	Display management class header file	223
HumSensor.cpp	Defines function of class HumSensor	226
HumSensor.h	Class HumSensor header file	227
I2C.cpp	Two-wire interface (I2C) source file	227
I2C.h	I2C class header file	228
int.cpp	Interrupt management source file	231
keepAliveLed.cpp	Definition of function for class keepAliveLed	234
keepAliveLed.h	Class keepAliveLed header file	235
LCD.cpp	LCD class source file	236
LCD.h	LCD class header file	237
LinkedList.cpp	Linked List library source file	249
LinkedList.h	Linked List library header file	249
main.cpp	Background task file	250
main.h	Background task header file	253
operators.cpp	C++ operators definitions	254
operators.h	C++ operators definitions header file	255
scheduler.cpp	Defines scheduler class	257
scheduler.h	Scheduler class header file	258
Sensor.cpp	Sensor class source code file	260
Sensor.h	Sensor class header file	261
sensor_configuration.cpp	Sensor configuration file	262
sensor_configuration.h	Sensors configuration header file	264
SensorManagement.cpp	SensorManagement class source code file	265
SensorManagement.h	SensorManagement class header file	266
String.cpp	String class source file	268
String.h	String class header file	268

TempSensor.cpp	Defines function of class TempSensor	269
TempSensor.h	Class TempSensor header file	270
timer.cpp	Defines function for class timer	271
timer.h	Timer class header file	272
usart.cpp	BSW library for USART	273
usart.h	Header file for USART library	274
Watchdog.cpp	Class Watchdog source code file	275
Watchdog.h	Class Watchdog header file	276

Chapter 4

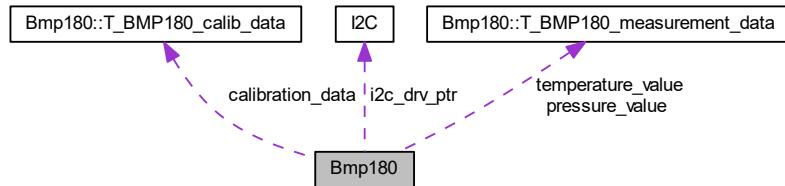
Class Documentation

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.

Private Types

- enum `T_BMP180_status` { `OK`, `COMM_FAILED` }
Enumeration defining the possible statuses for BMP180 sensor driver.
- enum `T_BMP180_conversion_type` { `NONE`, `TEMPERATURE`, `PRESSURE` }
Enumeration defining the different possible types of conversion.

Private Member Functions

- void `readCalibData` ()
Calibration data reading function.
- void `readChipID` ()
Chip ID read function.
- void `startNewTemperatureConversion` ()
Starts a new temperature conversion.
- void `CalculateTemperature` (`uint16_t` `UT`)
Temperature calculation function.

Private Attributes

- `I2C * i2c_drv_ptr`
- `uint8_t chip_id`
- `T_BMP180_status status`
- `T_BMP180_calib_data calibration_data`
- `T_BMP180_measurement_data temperature_value`
- `T_BMP180_measurement_data pressure_value`
- `T_BMP180_conversion_type conversionInProgress`

4.1.1 Detailed Description

BMP180 sensor class definition.

This class manages BMP180 driver.

Definition at line 41 of file Bmp180.h.

4.1.2 Member Enumeration Documentation

4.1.2.1 `T_BMP180_conversion_type`

```
enum Bmp180::T_BMP180_conversion_type [private]
```

Enumeration defining the different possible types of conversion.

Enumerator

NONE	No conversion is in progress
TEMPERATURE	A temperature conversion is in progress
PRESSURE	A pressure conversion is in progress

Definition at line 130 of file Bmp180.h.

4.1.2.2 T_BMP180_status

```
enum Bmp180::T_BMP180_status [private]
```

Enumeration defining the possible statuses for BMP180 sensor driver.

Enumerator

OK	
COMM_FAILED	

Definition at line 83 of file Bmp180.h.

4.1.3 Constructor & Destructor Documentation

4.1.3.1 Bmp180()

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

[Bmp180](#) class constructor.

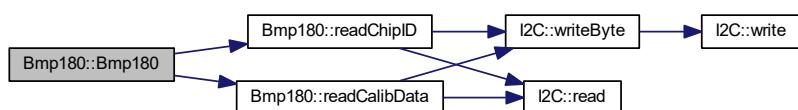
This function initializes the class [Bmp180](#). It reads the chip ID and reads calibration data. If the chip ID is incorrect, the status is set to communication failed.

Returns

Nothing

Definition at line 23 of file Bmp180.cpp.

Here is the call graph for this function:



4.1.4 Member Function Documentation

4.1.4.1 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	UT	Raw temperature value
----	----	-----------------------

Returns

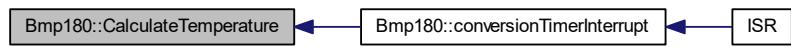
Nothing

Definition at line 220 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.4.2 conversionTimerInterrupt()

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

End of conversion interrupt.

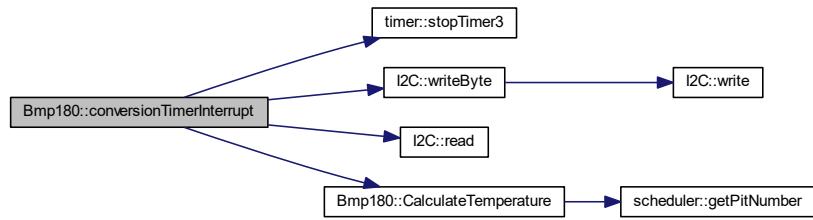
This function is called by the timer interrupt at the end of the conversion. It will retrieve the raw temperature or pressure value according to the conversion type and then calculate true value.

Returns

Nothing

Definition at line 171 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.4.3 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

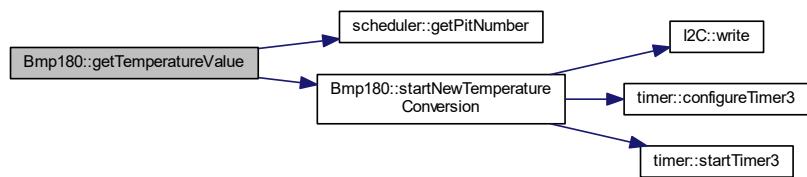
in	<i>data</i>	Pointer the location where the temperature data shall be stored.
----	-------------	--

Returns

True if a temperature measurement is available, false otherwise.

Definition at line 125 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.4.4 `readCalibData()`

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

Calibration data reading function.

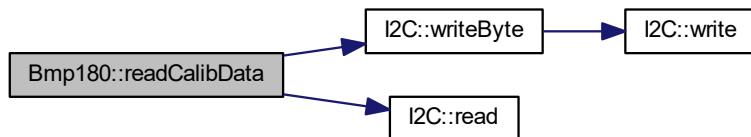
This function reads the pressure and temperature calibration data in BMP180 EEPROM using [I2C](#) bus. It also updates the driver status if the communication with the device is failed.

Returns

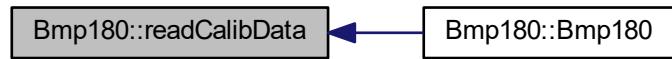
Nothing.

Definition at line 60 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.4.5 `readChipID()`

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

Chip ID read function.

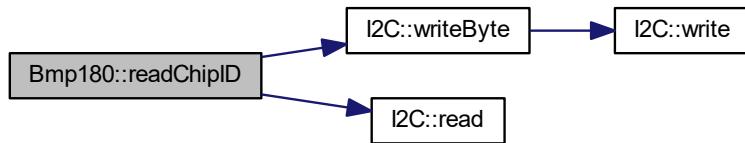
This function reads the ID of the sensor chip using `I2C` bus. It also updates the driver status if the communication is failed.

Returns

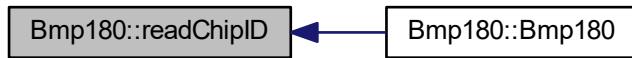
Nothing

Definition at line 106 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.4.6 `startNewTemperatureConversion()`

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

Starts a new temperature conversion.

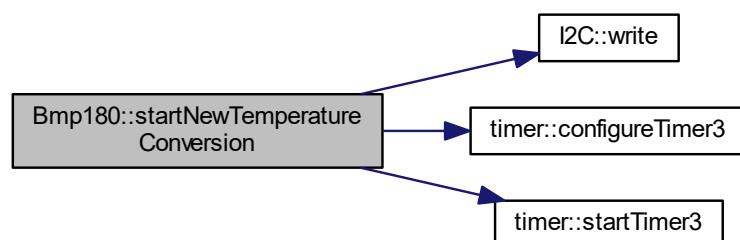
This function starts a new temperature conversion by writing 0x2E into register 0xF4 of sensor. It also starts a timer to retrieve sensor data after the conversion time.

Returns

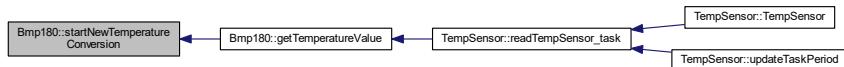
Nothing

Definition at line 150 of file `Bmp180.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.5 Member Data Documentation

4.1.5.1 calibration_data

`T_BMP180_calib_data Bmp180::calibration_data [private]`

Calibration data of the sensor

Definition at line 111 of file Bmp180.h.

4.1.5.2 chip_id

`uint8_t Bmp180::chip_id [private]`

`Sensor` chip ID

Definition at line 78 of file Bmp180.h.

4.1.5.3 conversionInProgress

`T_BMP180_conversion_type Bmp180::conversionInProgress [private]`

Definition at line 138 of file Bmp180.h.

4.1.5.4 i2c_drv_ptr

`I2C* Bmp180::i2c_drv_ptr [private]`

Pointer to the `I2C` driver object

Definition at line 77 of file Bmp180.h.

4.1.5.5 pressure_value

`T_BMP180_measurement_data` `Bmp180::pressure_value` [private]

Pressure data structure

Definition at line 125 of file Bmp180.h.

4.1.5.6 status

`T_BMP180_status` `Bmp180::status` [private]

Sensor status

Definition at line 90 of file Bmp180.h.

4.1.5.7 temperature_value

`T_BMP180_measurement_data` `Bmp180::temperature_value` [private]

Temperature data structure

Definition at line 124 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.

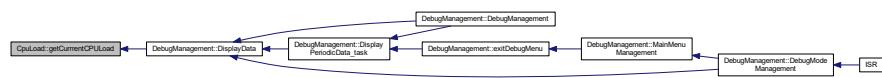
This function returns the current CPU load value

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.

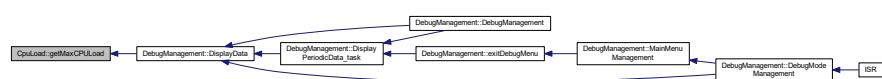
This function returns the maximum CPU load value

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:

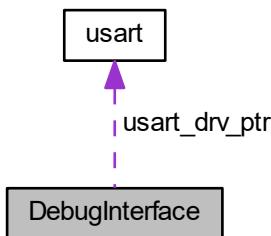
- [DebugManagement.h](#)

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.

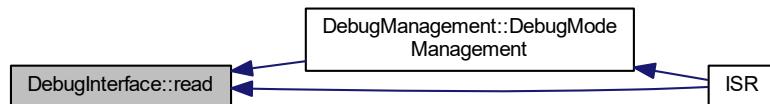
This function will read the last received byte on USART link

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

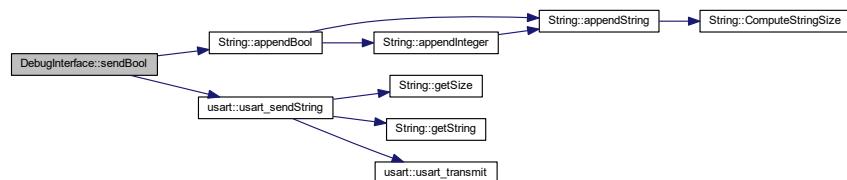
in	<code>data</code>	boolean data to be sent
in	<code>isText</code>	String conversion configuration

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

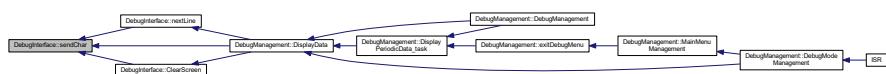
Nothing

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

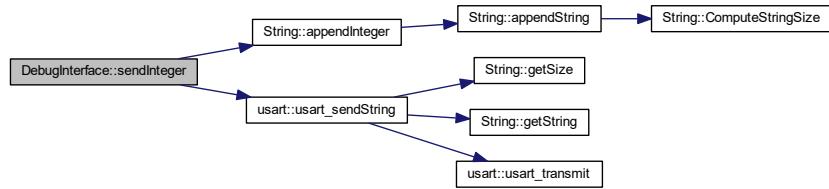
in	data	integer data to be sent
in	base	numerical base used to convert integer into string (between 2 and 36)

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

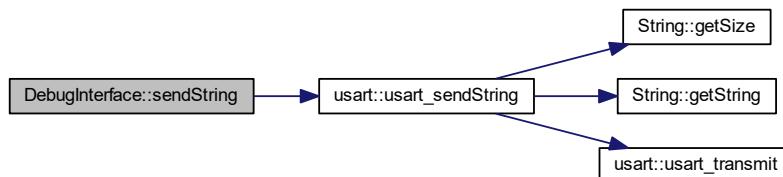
in	<i>str</i>	Pointer to the string being sent
----	------------	----------------------------------

Returns

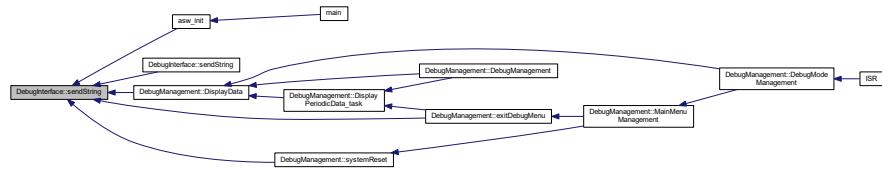
Nothing

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

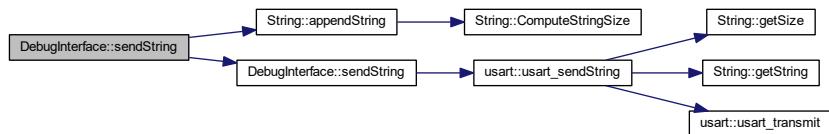
in	<i>str</i>	Pointer to the chain to send.
----	------------	-------------------------------

Returns

Nothing

Definition at line 37 of file DebugInterface.cpp.

Here is the call graph for this function:



4.4.4 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:

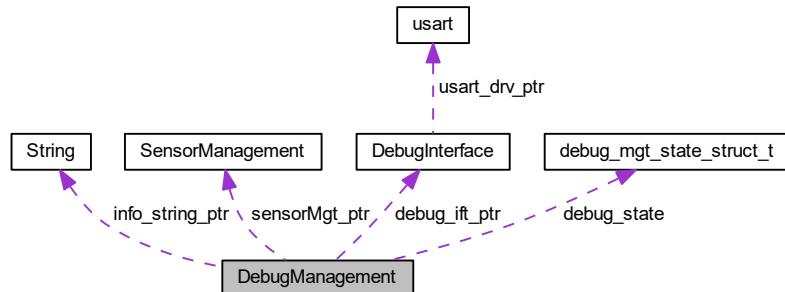
- [DebugInterface.h](#)
- [DebugInterface.cpp](#)

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 * getIfitPtr \(\)](#)
Interface pointer get function.
- [uint8_t * getMenuStringPtr \(\)](#)
Menu string get function.
- [String * getInfoStringPtr \(\)](#)
Info string get function.
- [void setInfoStringPtr \(String *addr\)](#)
Info message setting function.

Static Public Member Functions

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

Displays periodic data on usart link.

Private Member Functions

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

Private Attributes

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

4.5.1 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.

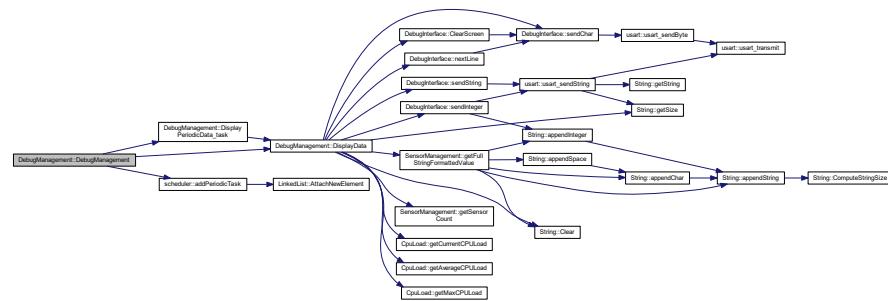
This function initializes the class. If needed, it creates a new instance of debug interface object.

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 :

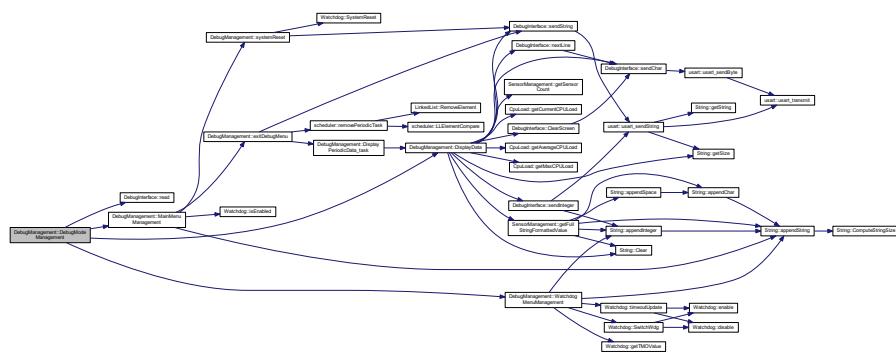
- MAIN_MENU state : handles user choice in main menu and selects next state
- WDG_MENU state : handles user choice in watchdog menu and selects next state
It is called each time a data is received on USART and debug mode is active.

Returns

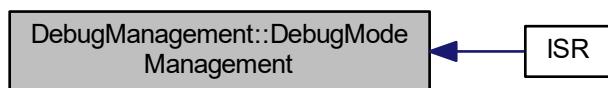
True if the debug mode shall be closed, false otherwise

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.

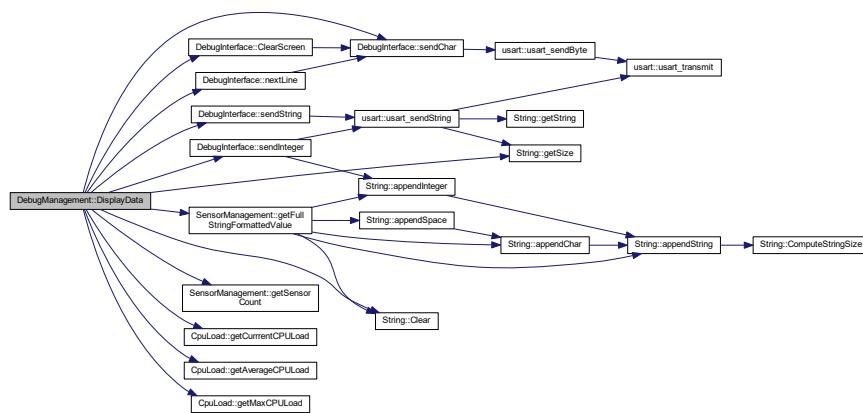
This task displays the menu and periodic data (temperature, humidity and CPU load) on usart screen.

Returns

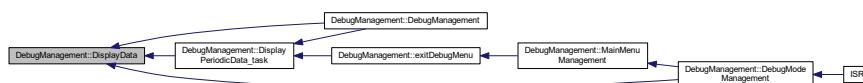
Nothing

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.

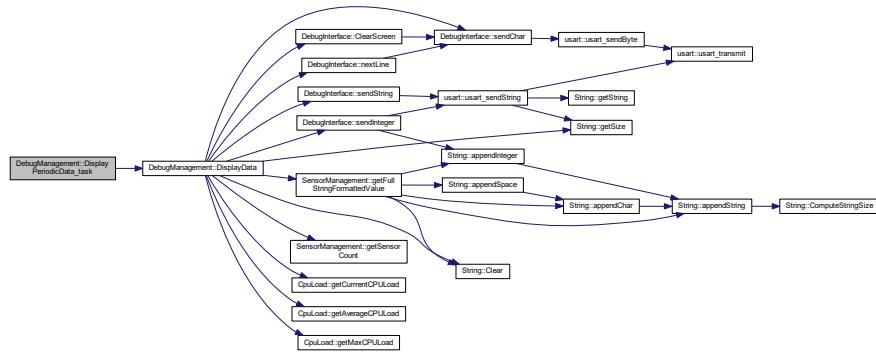
This task displays the menu and periodic data (temperature, humidity and CPU load) on uart screen. It only calls the function DisplayData.

Returns

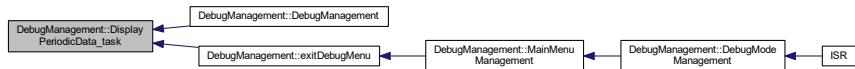
Nothing

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.

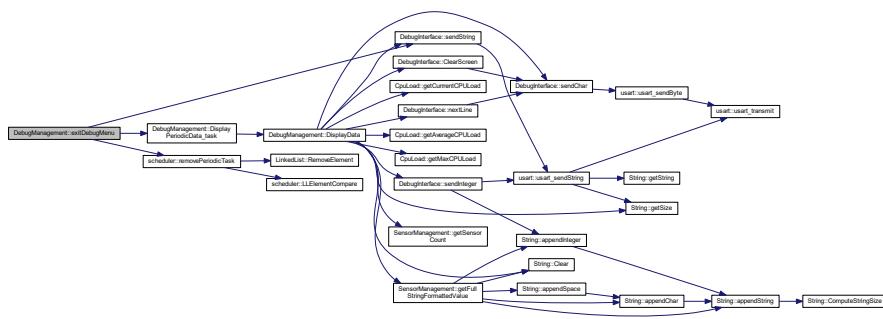
This function prepares the exit of debug menu. It writes the message "Bye !" on the screen and removes the periodic task from the scheduler.

Returns

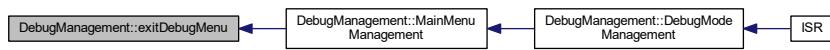
Nothing.

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::getIfptPtr ( ) [inline]
```

Interface pointer get function.

This function returns the pointer to the debug interface object

Returns

Pointer to debug interface

Definition at line 95 of file DebugManagement.h.

4.5.3.6 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

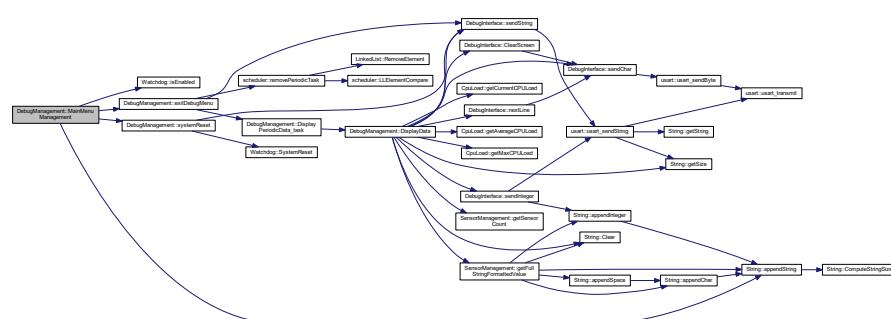
in	<i>rcv_char</i>	Character received on USART bus.
----	-----------------	----------------------------------

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.

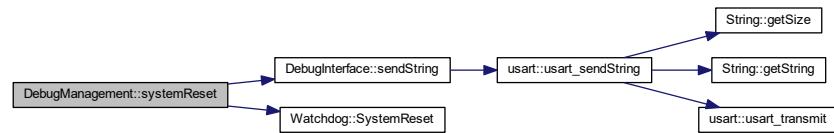
This function provokes a reset of the system. It displays a message on the screen and calls the reset function from watchdog class.

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

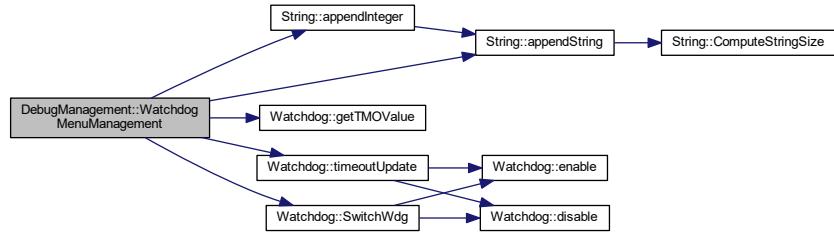
in	<code>rcv_char</code>	Character received on USART bus.
----	-----------------------	----------------------------------

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:

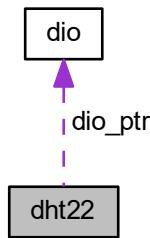
- [DebugManagement.h](#)
- [DebugManagement.cpp](#)

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

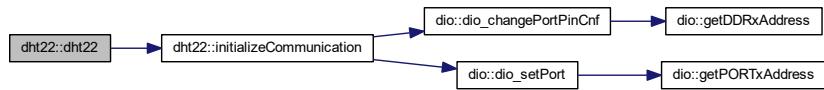
in	<i>port</i>	Encoded configuration of the port used for 1-wire communication.
----	-------------	--

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

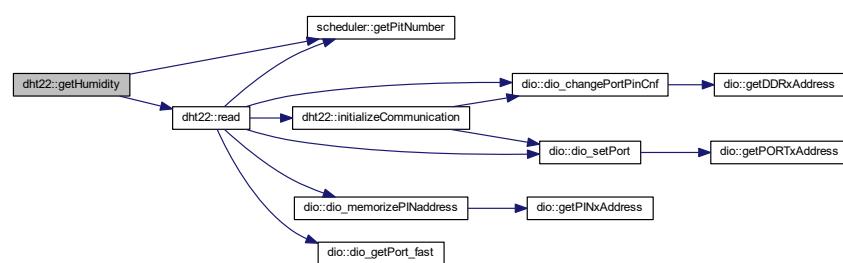
in	<i>humidity</i>	Address where the humidity shall be written
----	-----------------	---

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

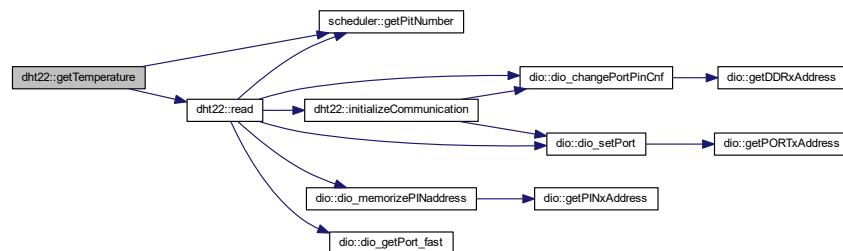
in	<i>temperature</i>	Address where the temperature shall be written
----	--------------------	--

Returns

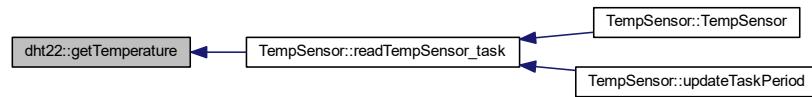
Validity of the data

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.

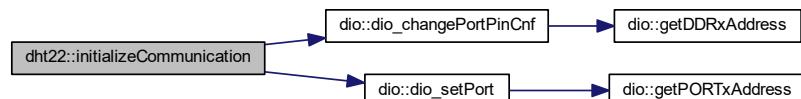
This function initializes the communication with DHT22 using 1-wire protocol

Returns

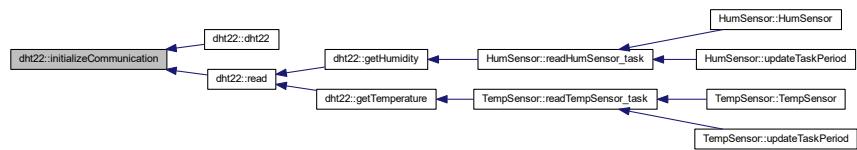
Nothing

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.

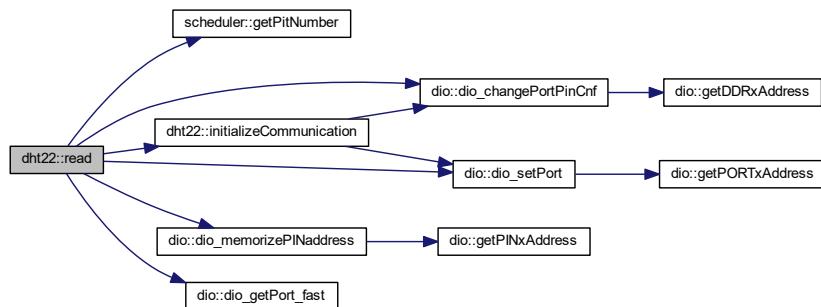
This function communicates with DHT22 using 1-wire protocol to read raw values of temperature and humidity. A checksum check is done when communication is finished to validate the received data. Validity of the data, temperature and humidity values are memorized in the associated class members.

Returns

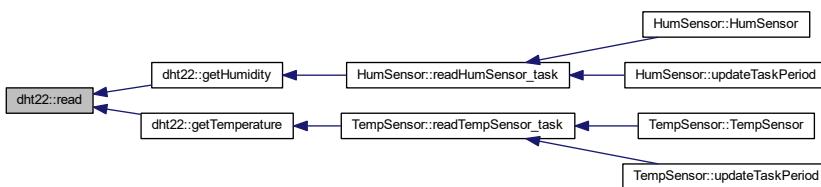
Nothing

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

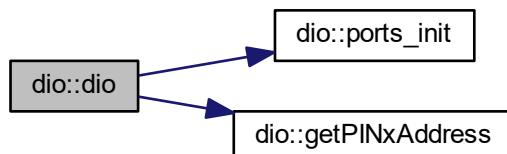
Initializes class `dio` and calls DIO hardware initialization function

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

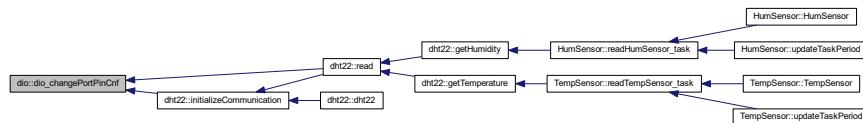
Nothing

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.

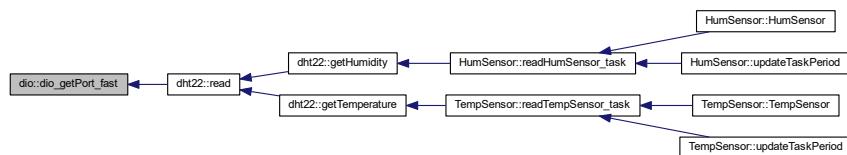
This function gets the logical value of the memorized pin. The corresponding port and pin index are stored in class members PINx_addr_mem and PINx_idx_mem. This mechanism is used to speed up reading time as this function no longer needs to extract register address and pin index from portcode.

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

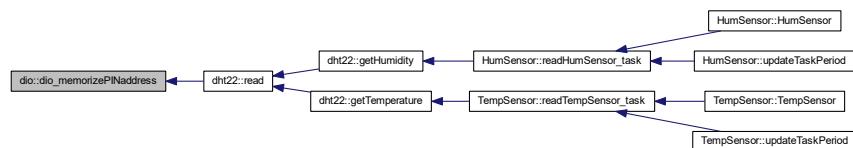
Nothing

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

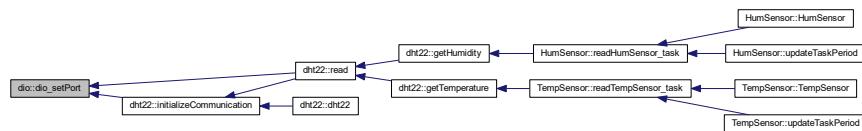
Nothing

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

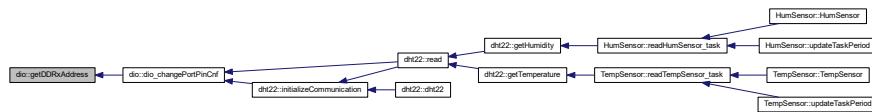
in	<i>portcode</i>	Encoded port code
----	-----------------	-------------------

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 PINx where x is encoded into the parameter portcode.

Parameters

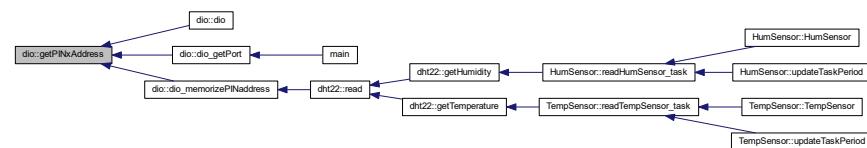
in	<i>portcode</i>	Encoded port code
----	-----------------	-------------------

Returns

Pointer to the PINx register

Definition at line 54 of file dio.cpp.

Here is the caller graph for this function:



4.7.3.9 getPORTxAddress()

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

Gets the physical address of the requested register PORTx.

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

Parameters

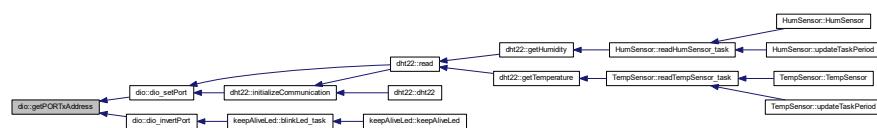
in	<i>portcode</i>	Encoded port code
----	-----------------	-------------------

Returns

Pointer to the PORTx register

Definition at line 25 of file dio.cpp.

Here is the caller graph for this function:

**4.7.3.10 ports_init()**

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

Digital ports hardware initialization function.

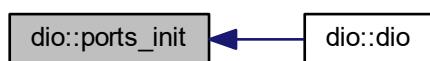
This function initializes digital ports as input or output and sets their initial values

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:

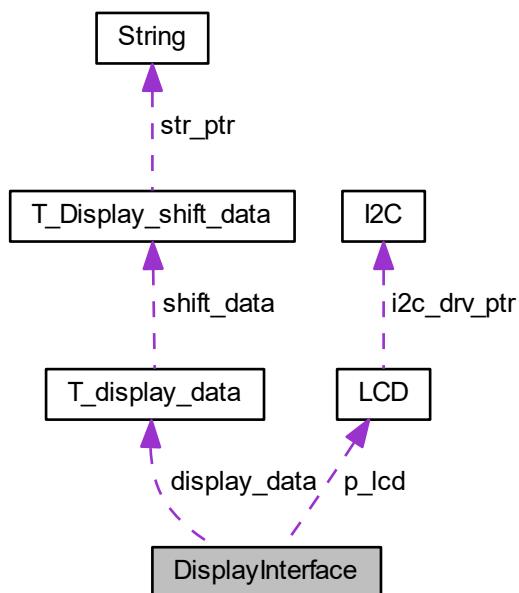
- [dio.h](#)
- [dio.cpp](#)

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

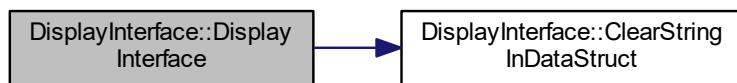
in	<i>LCD_init_cnf</i>	Initial configuration of the screen
----	---------------------	-------------------------------------

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.

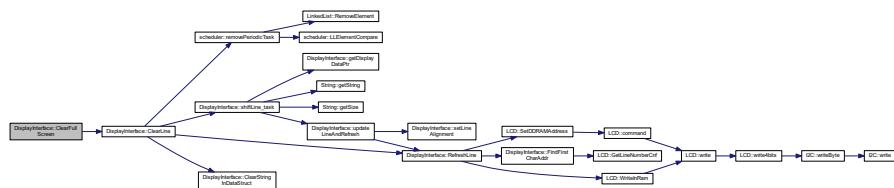
This function clears the entire display. It uses the ClearLine function on every line of the screen.

Returns

Nothing

Definition at line 269 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.3.2 ClearLine()

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

Line clearing function.

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

Parameters

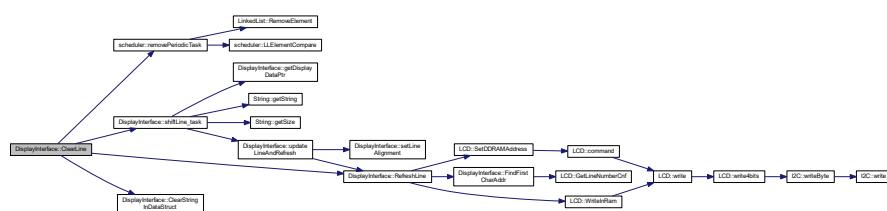
in	<i>line</i>	Line to clear
----	-------------	---------------

Returns

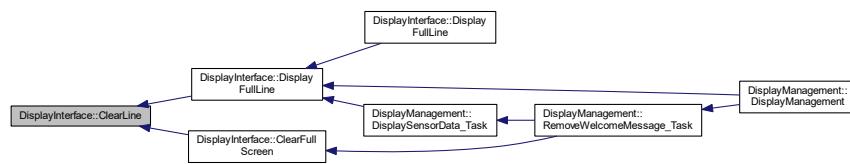
True if the line has been cleared, false otherwise

Definition at line 224 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.3.3 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

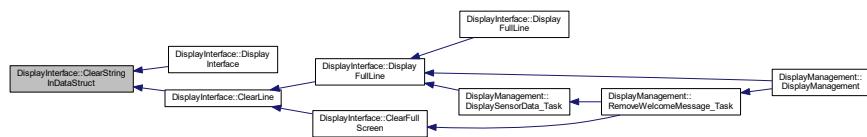
in	line	Line to clear
----	------	---------------

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

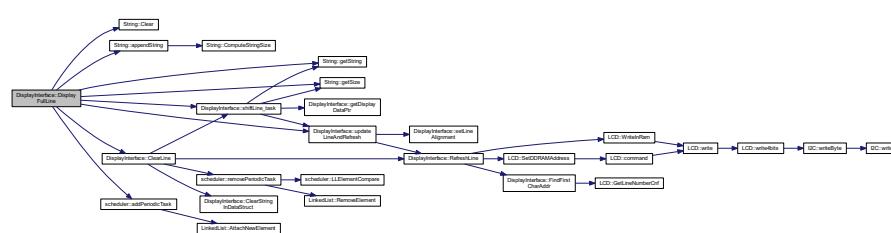
in	str	Pointer to the string to display
in	size	Size of the string to display
in	line	Index of the line where the string shall be displayed
in	mode	Display mode
in	alignment	Requested alignment for the line

Returns

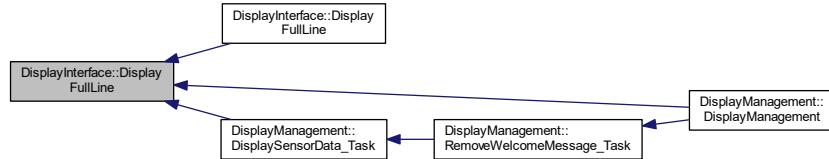
True if the line has been correctly displayed, false otherwise

Definition at line 59 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



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

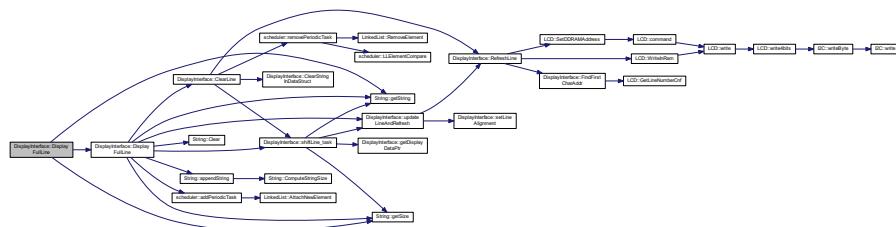
in	<i>str</i>	Pointer to the string to display
in	<i>line</i>	Index of the line where the string shall be displayed
in	<i>mode</i>	Display mode
in	<i>alignment</i>	Requested alignment for the line

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

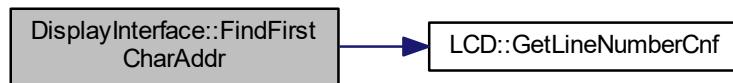
<code>in</code>	<code>line</code>	Line which address shall be found
-----------------	-------------------	-----------------------------------

Returns

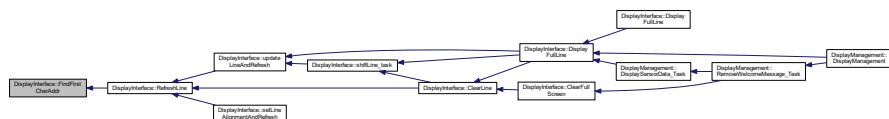
Address in DDRAM of the first character of the line

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.

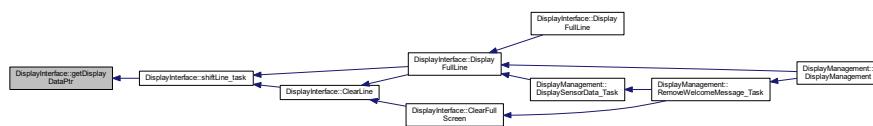
This function returns a pointer to the display data structure.

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

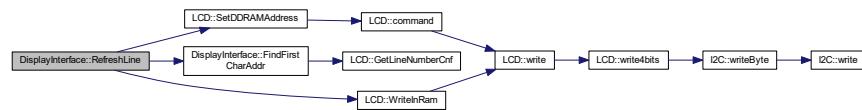
in	<i>line</i>	Line to refresh
----	-------------	-----------------

Returns

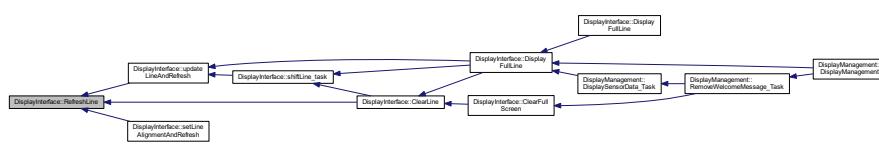
Nothing

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

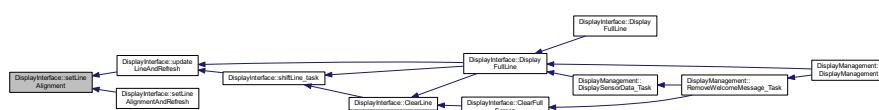
in	<i>line</i>	Line to update
----	-------------	----------------

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

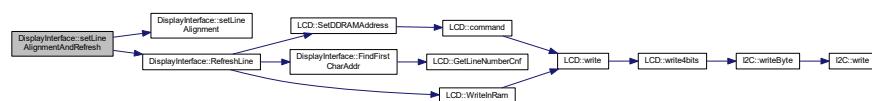
in	<i>line</i>	Requested line to update
in	<i>alignment</i>	Requested alignment for the text

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.

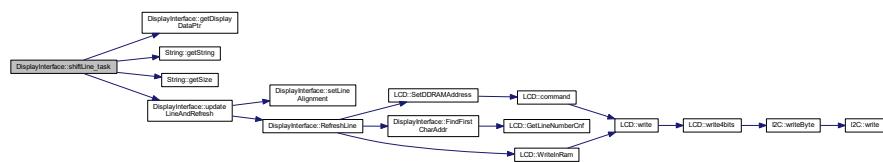
This function is called periodically by the scheduler. It shifts all the lines in line shifting mode and updates the data structures.

Returns

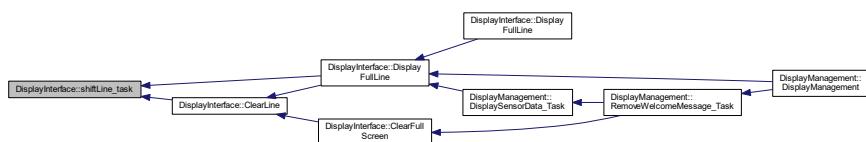
Nothing

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

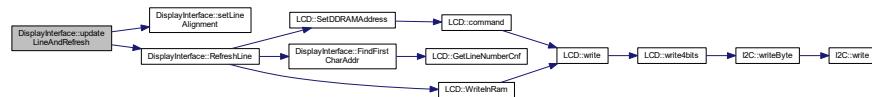
in	str	Pointer to the string to display
in	size	Size of the string
in	line	Line to update

Returns

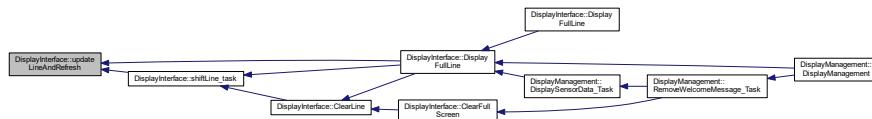
Nothing

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:

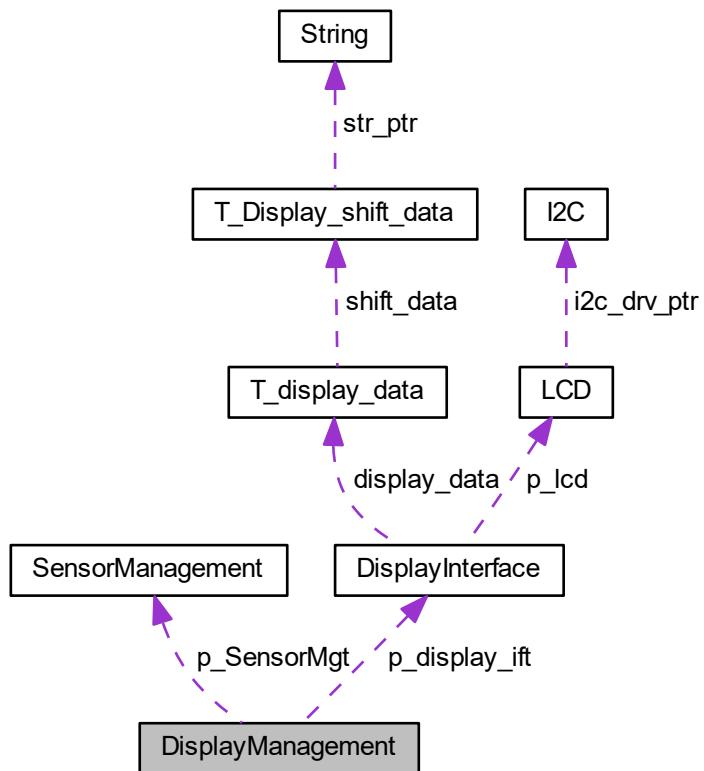
- [DisplayInterface.h](#)
- [DisplayInterface.cpp](#)

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.

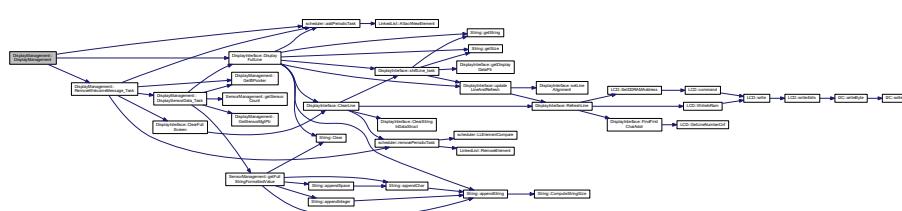
It created a display interface object and initializes all class variables.

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.

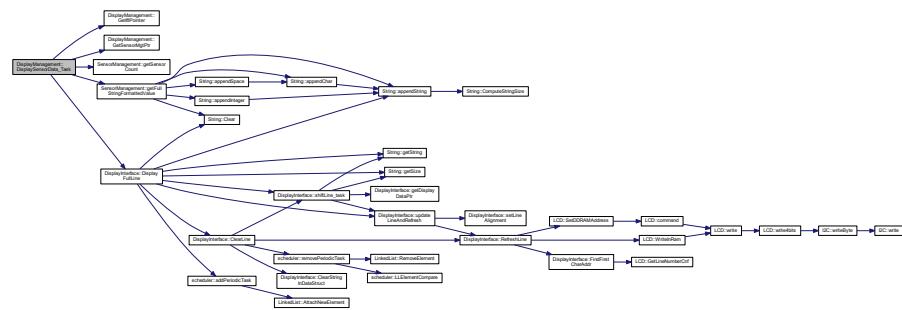
It is called periodically by scheduler.

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.

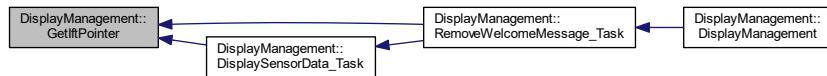
This function returns the pointer to the display interface object

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.

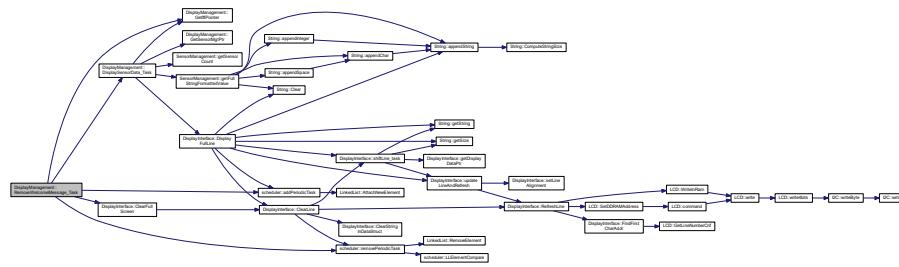
This task clears the welcome message from the screen and start periodic display of sensor data. This task shall be added in scheduler when the welcome message is displayed on screen. As it shall be called only once, the task removes itself from the scheduler after the first call.

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:

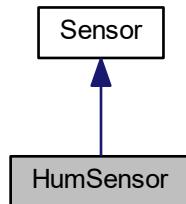
- [DisplayManagement.h](#)
- [DisplayManagement.cpp](#)

4.10 HumSensor Class Reference

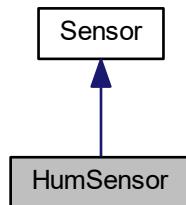
Class for humidity sensor.

```
#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.

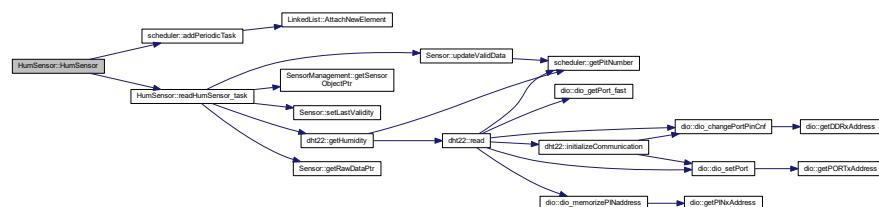
This function initializes all data of the class [HumSensor](#). If needed, it creates a new instance of the DHT22 sensor object. It also adds the periodic task in the scheduler.

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.

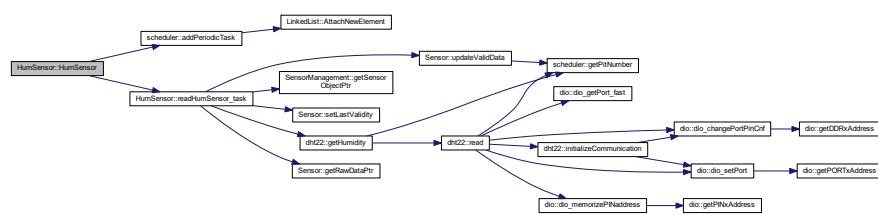
This function initializes all data of the class `HumSensor`. It sets validity timeout and task period to the given value. If needed, it creates a new instance of the DHT22 sensor object. It also adds the periodic task in the scheduler.

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.

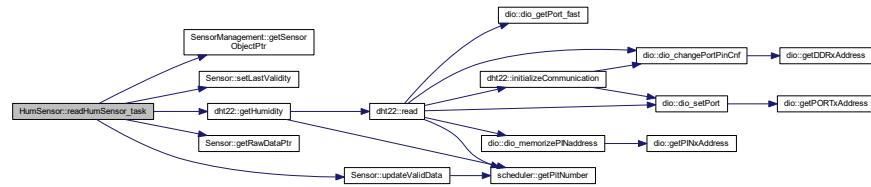
This task reads humidity data using DHT22 driver. It is called periodically.

Returns

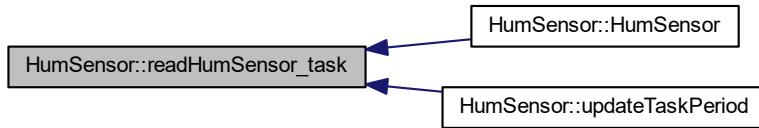
Nothing

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

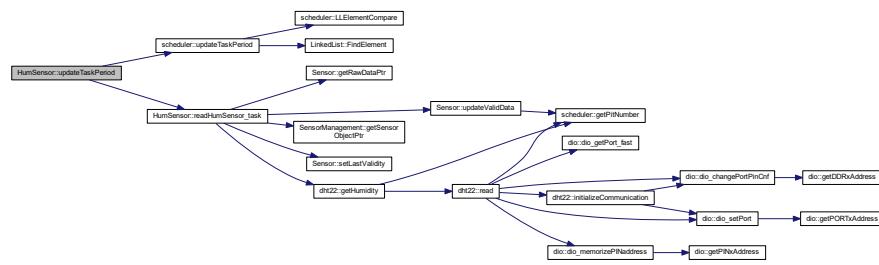
in	<i>period</i>	New period of the task
----	---------------	------------------------

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

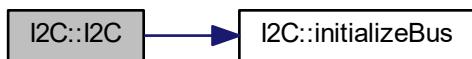
in	<i>l_bitrate</i>	Requested bitrate for I2C bus (in Hz)
----	------------------	---

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

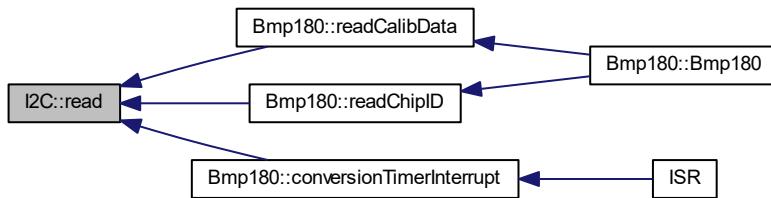
in	<i>i2c_address</i>	I2C address of the device
in	<i>size</i>	Number of bytes to read
out	<i>buf_ptr</i>	Pointer to the start of the reception buffer.

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

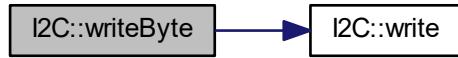
in	<i>data</i>	Data to send
in	<i>tx_address</i>	I2C address of the device
in	<i>sendStopCond</i>	Defines if the stop condition shall be sent or not

Returns

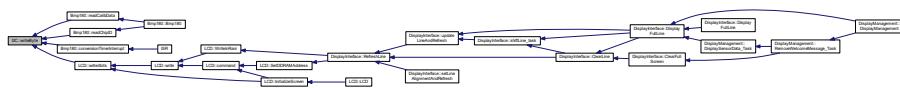
True if transmission is completed, False if an error has occurred

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

- [keepAliveLed \(\)](#)

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.

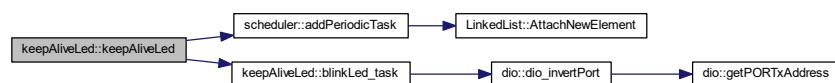
This function initializes the class `keepAliveLed`

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.

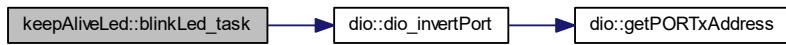
This function is inserted into the scheduler. It changes the state of the LED output to make it blink

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:

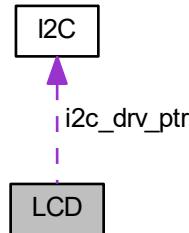
- [keepAliveLed.h](#)
- [keepAliveLed.cpp](#)

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

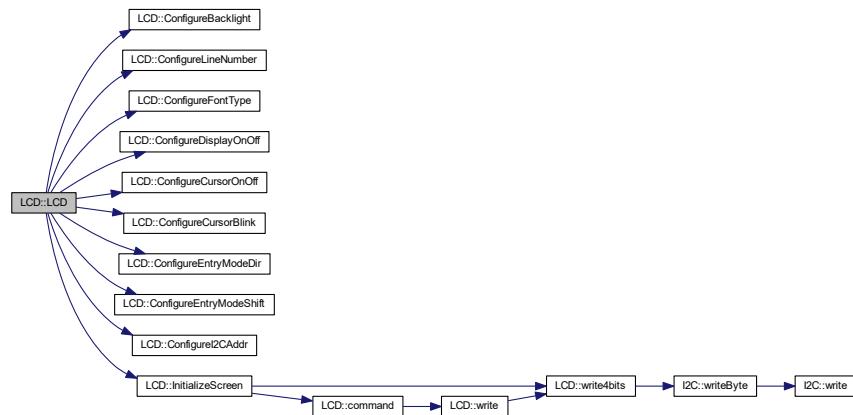
in	<code>init_conf</code>	Initial configuration structure
----	------------------------	---------------------------------

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

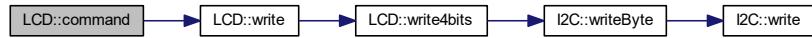
in	<i>cmd</i>	Requested command
----	------------	-------------------

Returns

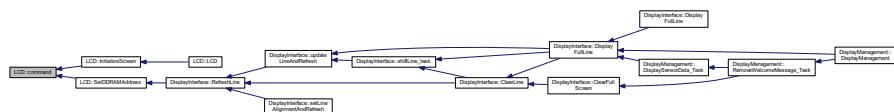
Nothing

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	<i>param</i>	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

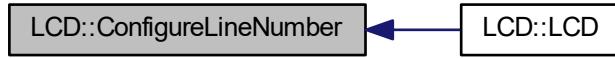
<code>in</code>	<code>param</code>	Configuration value
-----------------	--------------------	---------------------

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 ddrum_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.

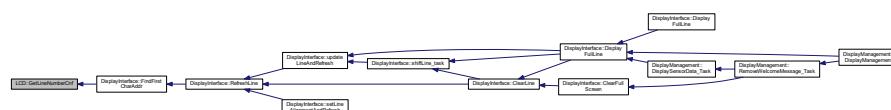
This function returns the line number configuration of the screen : 1 or 2 lines mode.

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

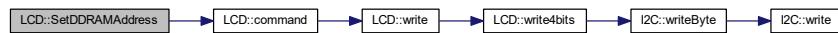
in	<i>addr</i>	New DDRAM address
----	-------------	-------------------

Returns

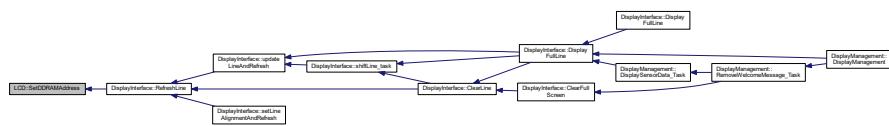
Nothing

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

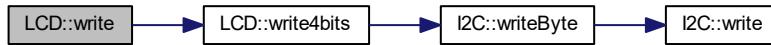
in	<i>data</i>	8-bit data for D0-7 pins of screen
in	<i>mode</i>	Requested mode for LCD communication

Returns

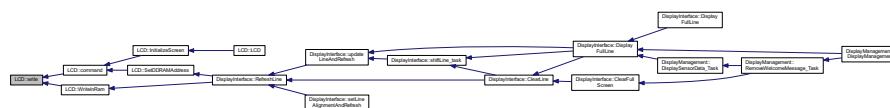
Nothing

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

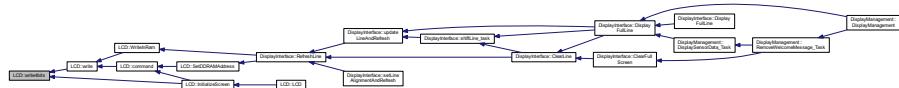
Nothing

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

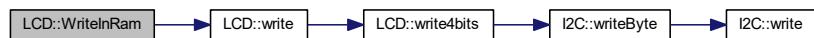
in	<i>a_char</i>	Data byte to write in RAM
in	<i>area</i>	Area in RAM where the data will be written : DDRAM or CGRAM

Returns

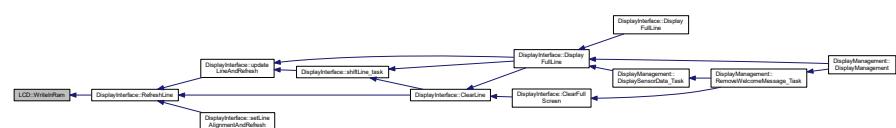
Nothing

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:

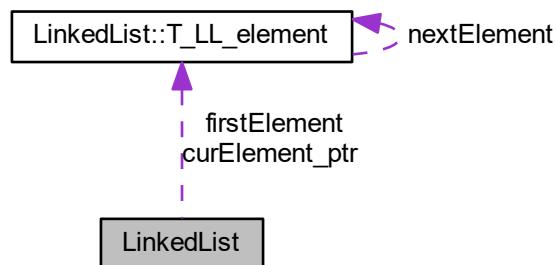
- [LCD.h](#)
- [LCD.cpp](#)

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

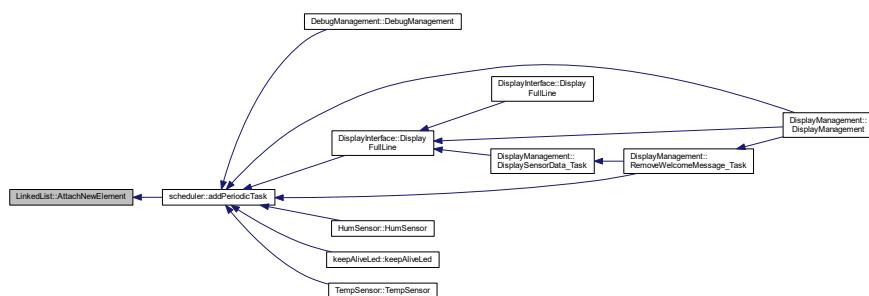
in	<i>data_ptr</i>	Pointer to the data element
----	-----------------	-----------------------------

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

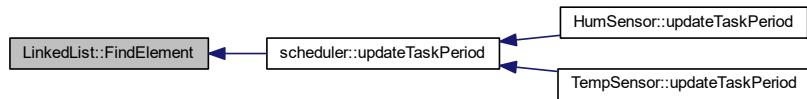
in	<i>comparisonFct_ptr</i>	Pointer to the comparison function
in	<i>reference_ptr</i>	Pointer to the element to find in the chain
out	<i>chainElement_ptr</i>	Pointer to pointer to the found element

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.

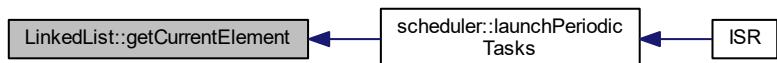
This function returns a pointer to the current pointed data in the chain.

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.

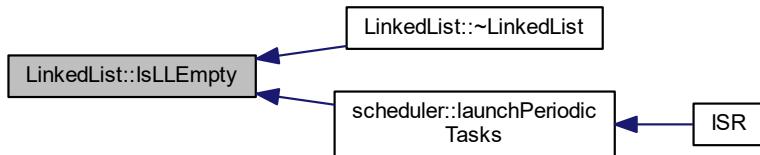
This function checks whether the linked list is empty or not (pointer to first element is equal to 0 or not).

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.

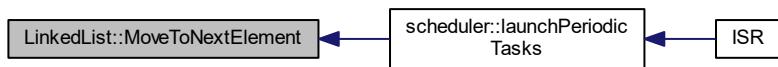
This function moves the element pointer to the next element of the chain.

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

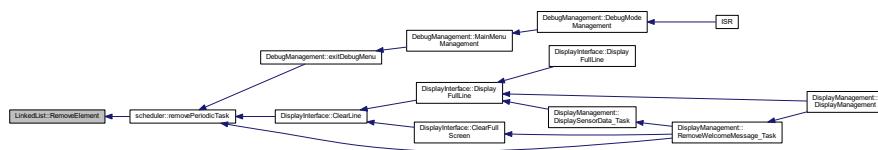
in	<i>comparisonFct_ptr</i>	Pointer to the comparison function to use
in	<i>reference_ptr</i>	Pointer to the reference data used for comparison

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:

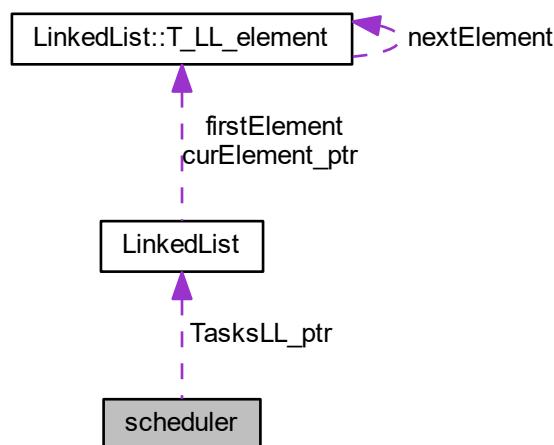
- [LinkedList.h](#)
- [LinkedList.cpp](#)

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

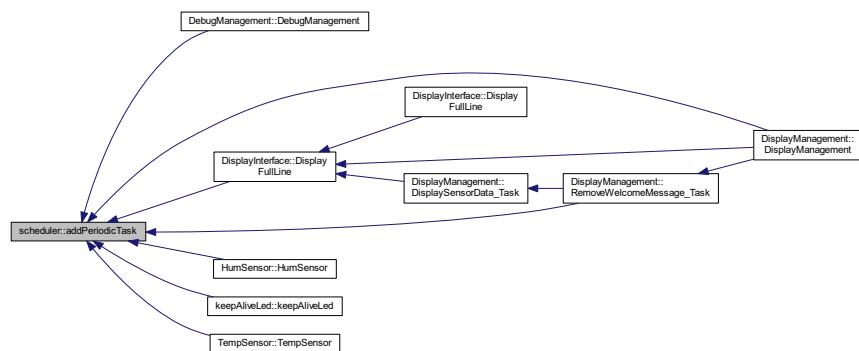
Nothing

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.

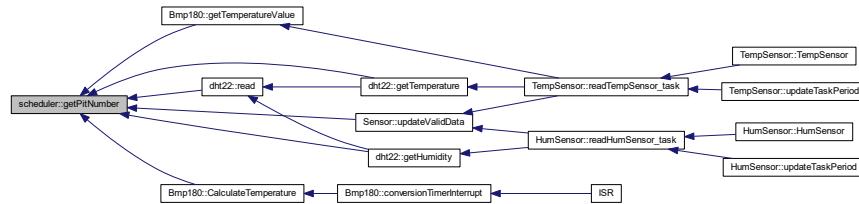
This function returns the PIT number

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.

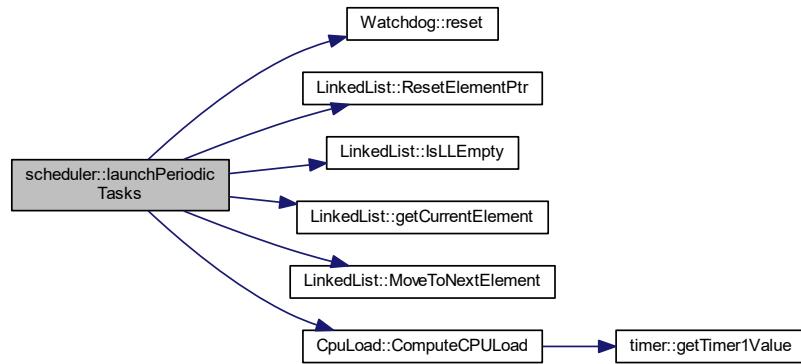
This function launches the scheduled tasks according to current software time and task configuration

Returns

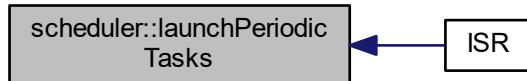
Nothing

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

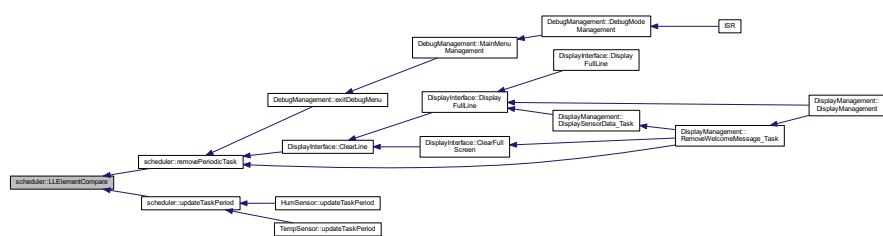
in	<i>LLElement</i>	Pointer to the linked list element
in	<i>CompareElement</i>	Pointer to the element to the compare

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

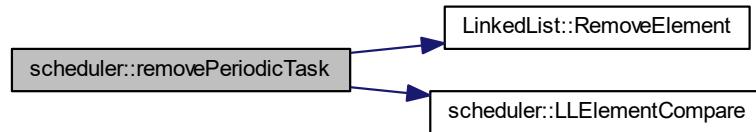
in	<i>task_ptr</i>	address of the task to remove from scheduler
----	-----------------	--

Returns

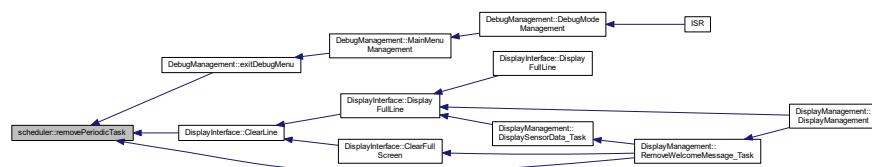
TRUE if the task has been removed, FALSE if the task does not exist in the scheduler

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

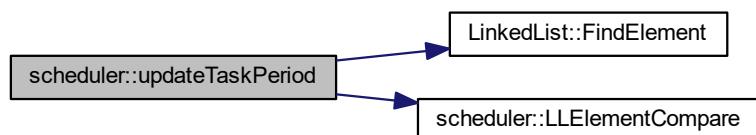
in	<i>task_ptr</i>	Pointer of the task to update
in	<i>period</i>	New period of the task

Returns

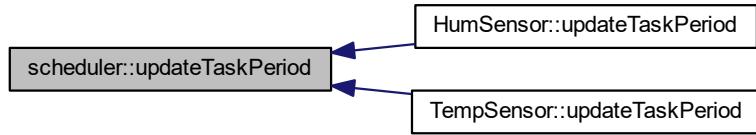
True if the update has been correctly done, false otherwise

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:

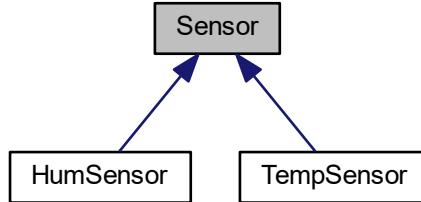
- [scheduler.h](#)
- [scheduler.cpp](#)

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.

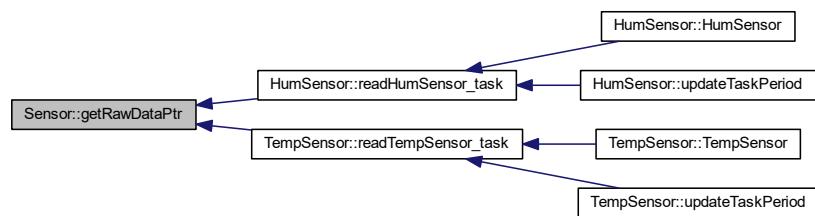
This function returns a pointer to the class member `raw_data`

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

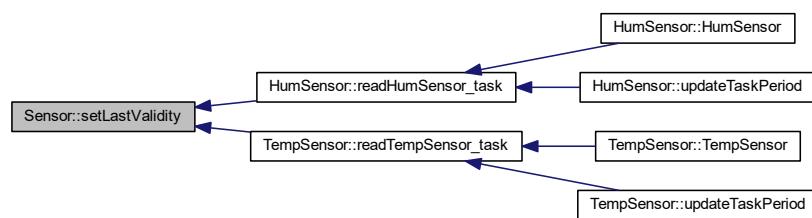
in	<i>validity</i>	Value of validity
----	-----------------	-------------------

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()

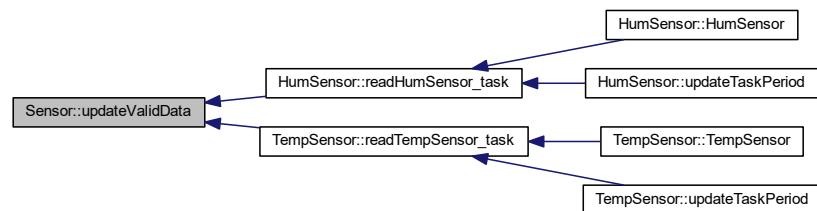
```
void Sensor::updateValidData ( )
```

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

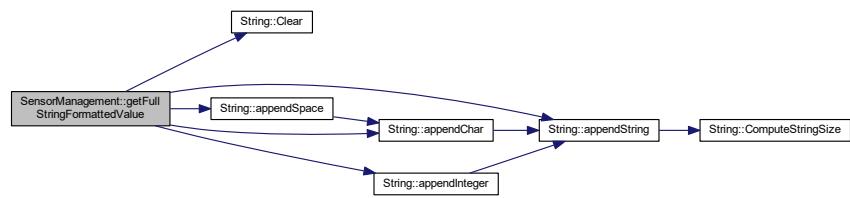
in	<i>sensor_idx</i>	Index of the requested sensor
out	<i>str</i>	Pointer to the formatted string

Returns

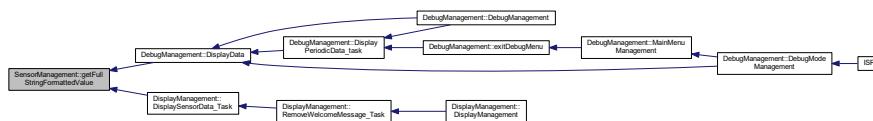
Nothing

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.

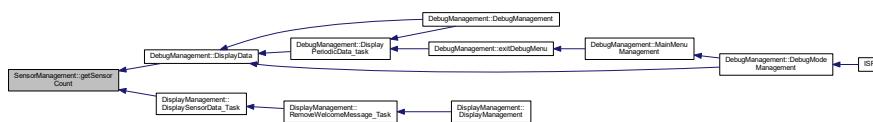
This function returns the number of sensors present in the SW

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

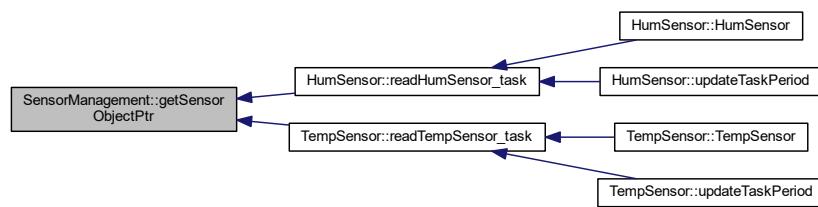
<code>in</code>	<code>type</code>	Type of sensor to find
-----------------	-------------------	------------------------

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

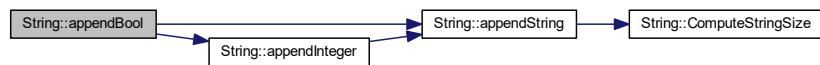
in	<code>data</code>	Boolean data to add
in	<code>isText</code>	Defines the conversion mode : text or integer

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

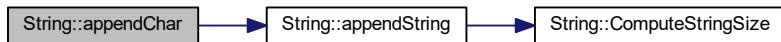
in	<i>data</i>	1-byte character to add
----	-------------	-------------------------

Returns

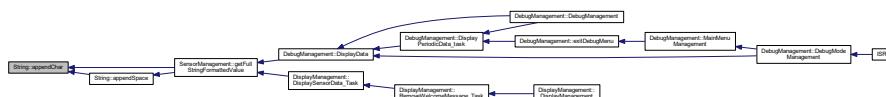
Nothing

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

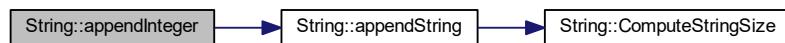
<code>in</code>	<code>value</code>	Integer to add
<code>in</code>	<code>base</code>	Base of computation of the integer (between 2 and 36)

Returns

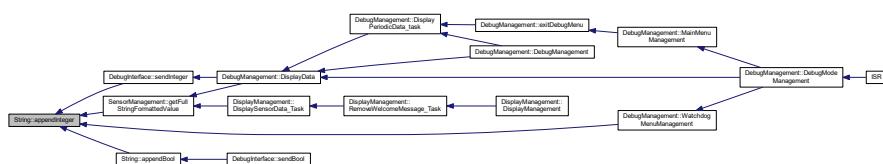
Nothing

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.

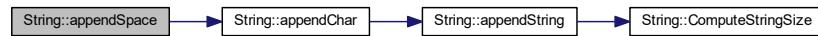
This function adds a space at the end of the string. It only calls appendChar function.

Returns

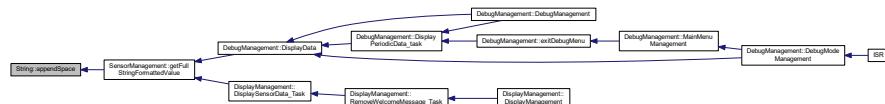
Nothing

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

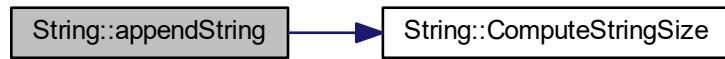
in	<i>str</i>	New string to add
----	------------	-------------------

Returns

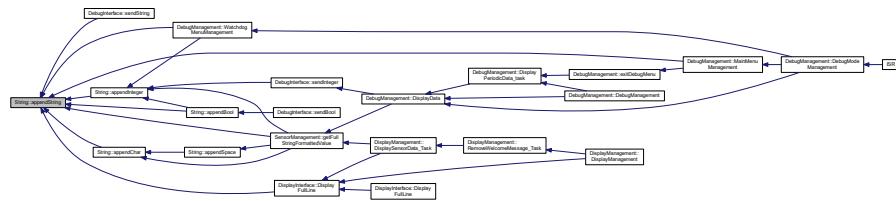
Nothing

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.

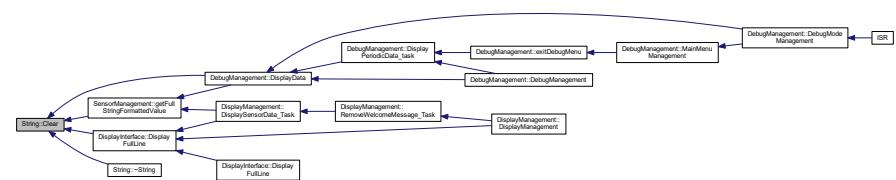
This function clears the string. Size is set to 0 and the memory is freed.

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 (
    uint8_t * str ) [private]
```

String size computation function.

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

Parameters

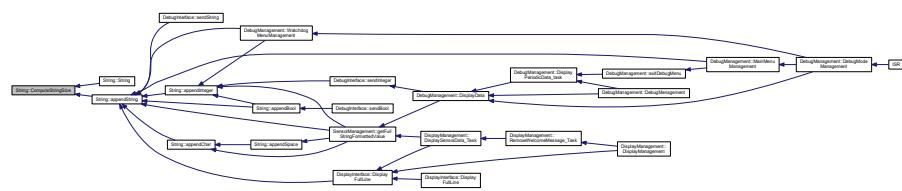
in	<i>str</i>	Pointer to the beginning of the string
----	------------	--

Returns

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

Definition at line 44 of file String.cpp.

Here is the caller graph for this function:



4.18.3.8 getSize()

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

Size get function.

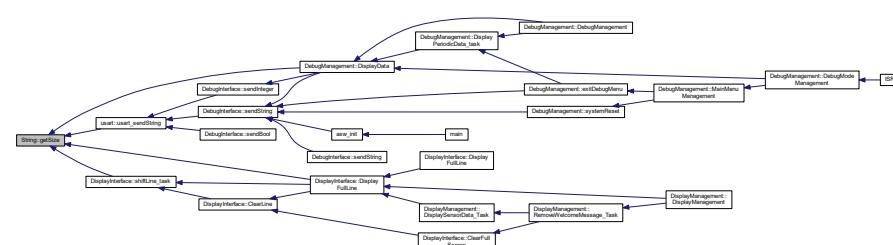
This function returns the size of the string.

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.

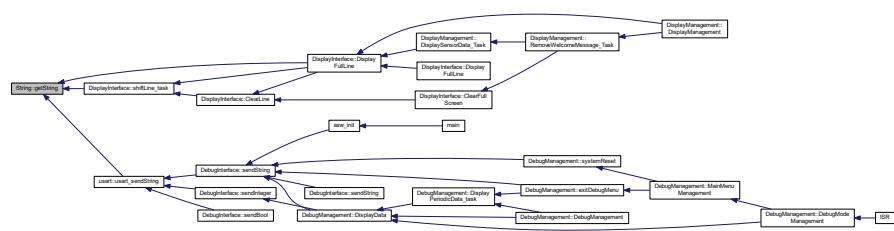
This function returns the pointer to the beginning of the string.

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 `isDebugActivated`
- 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 95 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 97 of file Bmp180.h.

4.20.2.2 AC2

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

Definition at line 98 of file Bmp180.h.

4.20.2.3 AC3

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

Definition at line 99 of file Bmp180.h.

4.20.2.4 AC4

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

Definition at line 100 of file Bmp180.h.

4.20.2.5 AC5

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

Definition at line 101 of file Bmp180.h.

4.20.2.6 AC6

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

Definition at line 102 of file Bmp180.h.

4.20.2.7 B1

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

Definition at line 103 of file Bmp180.h.

4.20.2.8 B2

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

Definition at line 104 of file Bmp180.h.

4.20.2.9 MB

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

Definition at line 105 of file Bmp180.h.

4.20.2.10 MC

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

Definition at line 106 of file Bmp180.h.

4.20.2.11 MD

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

Definition at line 107 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 116 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 119 of file Bmp180.h.

4.21.2.2 ts

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

Definition at line 120 of file Bmp180.h.

4.21.2.3 value

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

Definition at line 118 of file Bmp180.h.

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

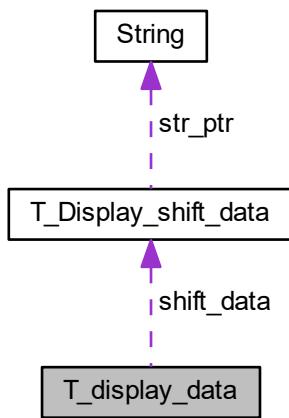
- [Bmp180.h](#)

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:

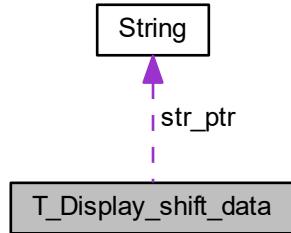
- [DisplayInterface.h](#)

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:

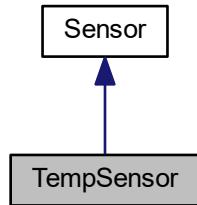
- [scheduler.h](#)

4.28 TempSensor Class Reference

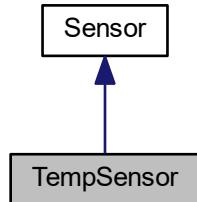
Class for temperature sensor.

```
#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.

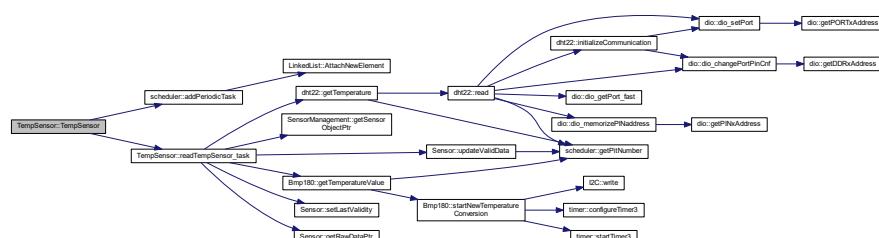
This function initializes all data of the class [TempSensor](#). If needed, it creates a new instance of the DHT22 and BMP180 sensors objects. It also adds the periodic task in the scheduler.

Returns

Nothing

Definition at line 29 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.

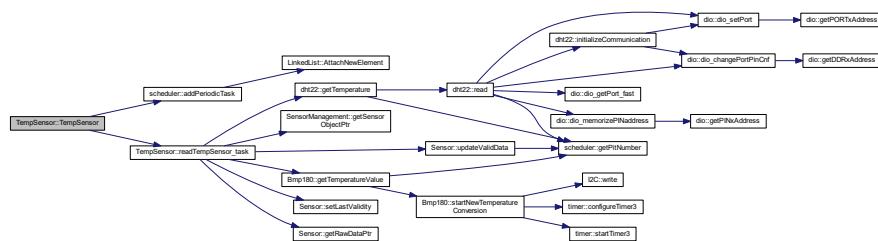
This function initializes all data of the class [TempSensor](#). It sets validity timeout and task period to the given value. If needed, it creates a new instance of the DHT22 and BMP180 sensor objects. It also adds the periodic task in the scheduler.

Returns

Nothing

Definition at line 44 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.

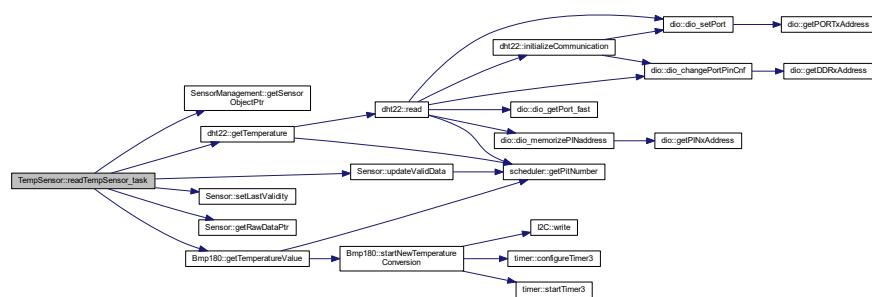
This task reads temperature data using DHT22 and BMP180 drivers. It is called periodically. The returned temperature is the mean between both sensors values, if only one sensor is valid, only this value is used .

Returns

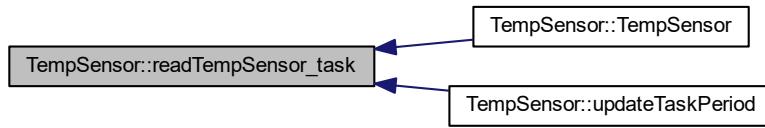
Nothing

Definition at line 58 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

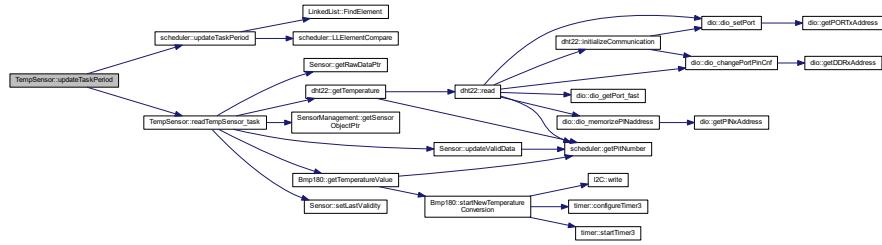
in *period* New period of the task

Returns

True if the period has been updated, false otherwise

Definition at line 111 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	<code>a_prescaler</code>	prescaler value
in	<code>a_ctcValue</code>	Value to which the counter will compare before raising an interrupt

Returns

Nothing

Definition at line 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.

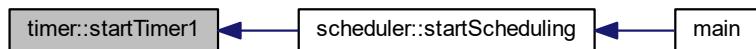
This functions starts Timer #1. Timer shall be initialized before this function is called.

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	<i>a_BaudRate</i>	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.

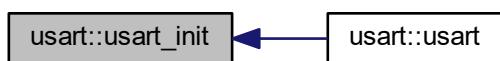
This function will initialize the USART using selected baudrate. User must pay attention to select one of the usually used Baud Rate (9600, 19200, 38400, 57600). Note that since an uint16 is used as argument, Baud rate cannot be more than 57600.

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

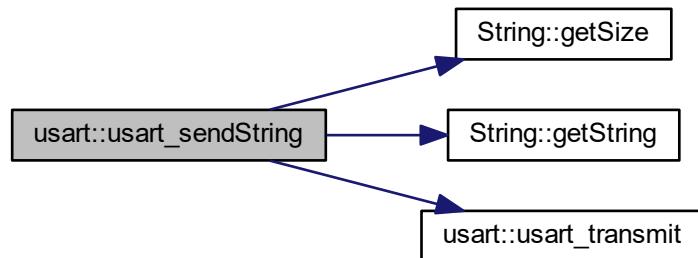
in	<i>str</i>	Pointer to the string being sent
----	------------	----------------------------------

Returns

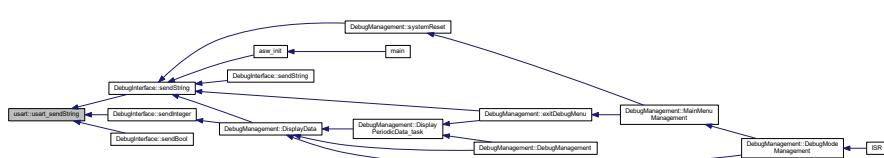
Nothing.

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

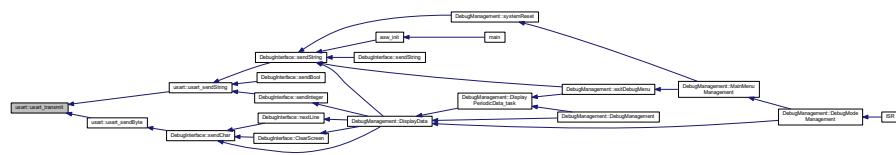
in	<i>Data</i>	Desired data char to transmit
----	-------------	-------------------------------

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.

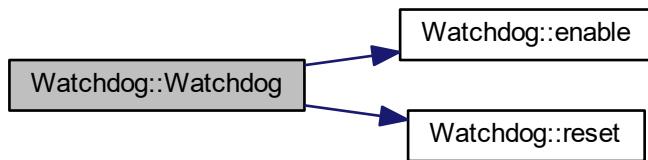
This function initializes the watchdog class. It enables the HW watchdog with a default timeout value.

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

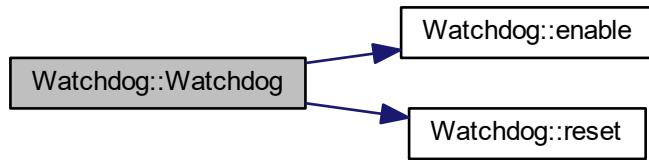
in	<i>timeout</i>	Timeout value requested for the watchdog
----	----------------	--

Returns

Nothing

Definition at line 26 of file Watchdog.cpp.

Here is the call graph for this function:



4.31.3 Member Function Documentation

4.31.3.1 disable()

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

[Watchdog](#) disabling function.

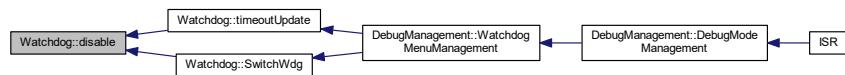
This function disables the watchdog by calling `wdt_disable` macro.

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

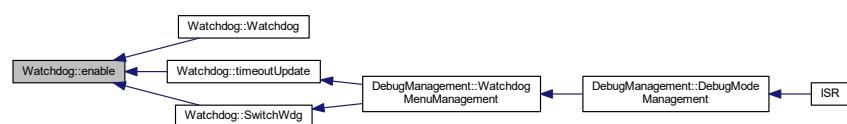
in	<i>value</i>	Timeout value
----	--------------	---------------

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.

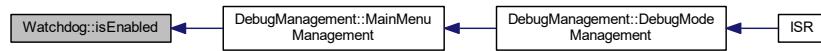
This function returns the current status of the watchdog : enabled or disabled.

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.

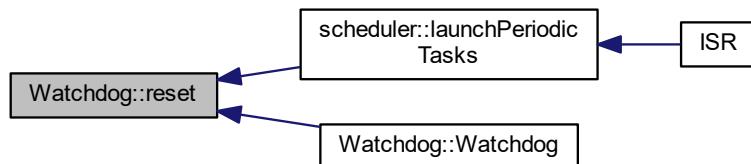
This function resets the watchdog timer by calling wdt_reset macro

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.

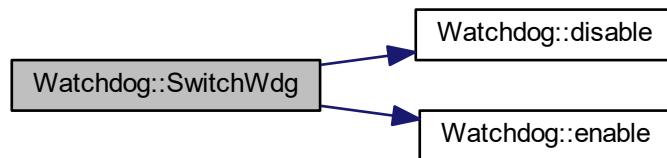
This function switches the state of the watchdog. If it was enabled, the function disables the watchdog, and if it was disabled, the function enables it with the memorized timeout value. The function returns the new status of the watchdog.

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.

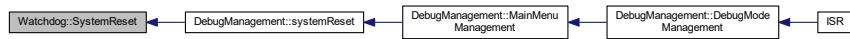
This function provokes a system reset by going in an infinite loop. Thus the watchdog will reset the CPU when the timeout occurs.

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

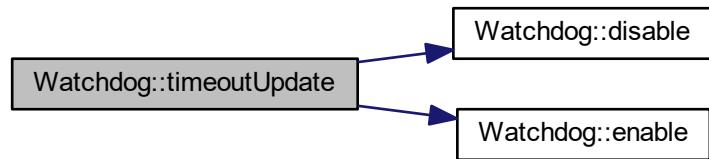
in	value	New timeout value
----	-------	-------------------

Returns

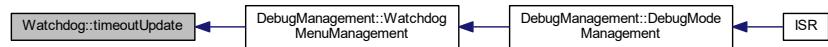
Nothing

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.

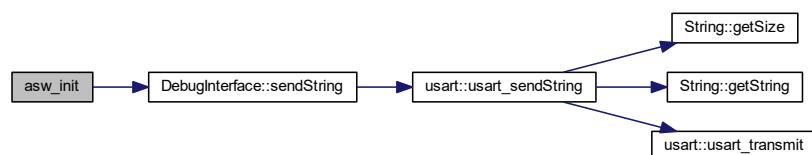
This function shall be called after BSW initialization function.

Returns

Nothing

Definition at line 35 of file asw.cpp.

Here is the call graph for this function:



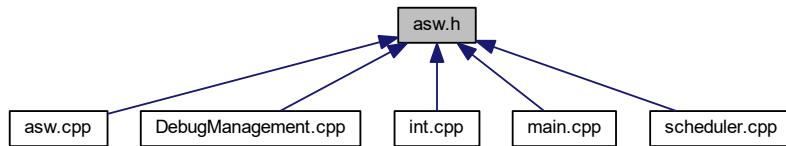
Here is the caller graph for this function:



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.

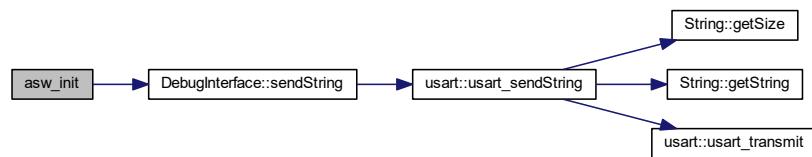
This function shall be called after BSW initialization function.

Returns

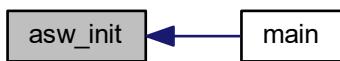
Nothing

Definition at line 35 of file asw.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



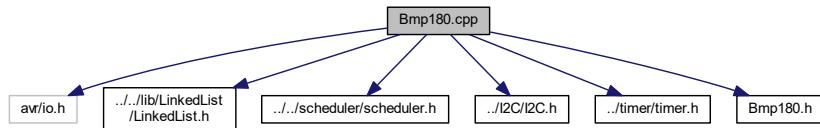
5.3 Bmp180.cpp File Reference

[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

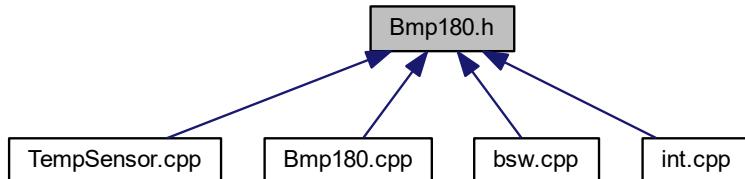
Pointer to BMP180 driver object

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_DATA_VALIDITY_TMO](#) 2
- #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

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_DATA_VALIDITY_TMO

```
#define BMP180_DATA_VALIDITY_TMO 2
```

Number of PITs after which the sensor value is declared invalid

Definition at line 23 of file Bmp180.h.

5.4.2.7 BMP180_I2C_ADDR

```
#define BMP180_I2C_ADDR 0x77
```

I2C address of the sensor

Definition at line 14 of file Bmp180.h.

5.4.2.8 BMP180_I2C_BITRATE

```
#define BMP180_I2C_BITRATE 100000
```

Bitrate used for I2C communication

Definition at line 13 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 28 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 29 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 26 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 25 of file Bmp180.h.

5.4.3 Variable Documentation

5.4.3.1 p_global_BSW_bmp180

```
Bmp180* p_global_BSW_bmp180
```

Pointer to BMP180 driver object

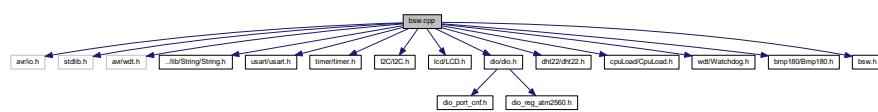
Definition at line 21 of file Bmp180.cpp.

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/usart.h"
#include "timer/timer.h"
#include "I2C/I2C.h"
#include "lcd/LCD.h"
#include "dio/dio.h"
#include "dht22/dht22.h"
#include "cpuLoad/CpuLoad.h"
#include "wdt/Watchdog.h"
#include "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.

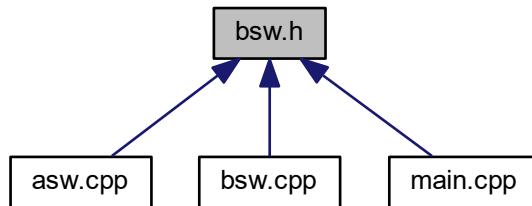
Here is the caller graph for this function:



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:

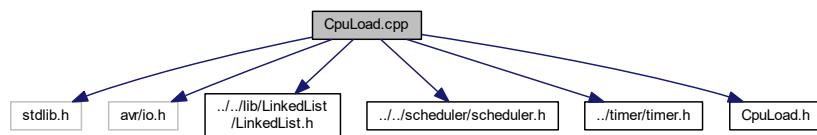


5.7 CpuLoad.cpp File Reference

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](#)

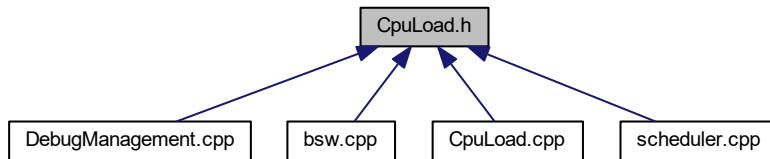
Pointer to cpu load library object

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

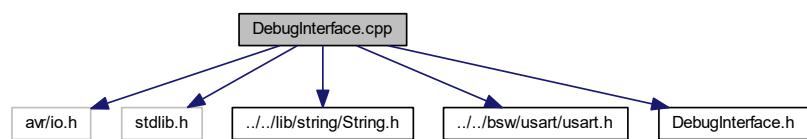
Definition at line 18 of file CpuLoad.cpp.

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
```

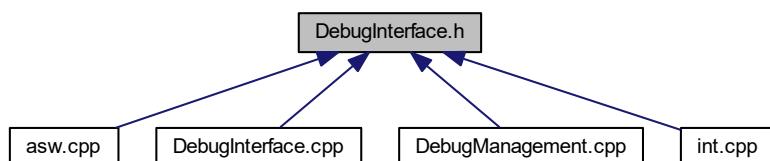
Pointer to USART debug interface object

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:

Include dependency graph: 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"
```

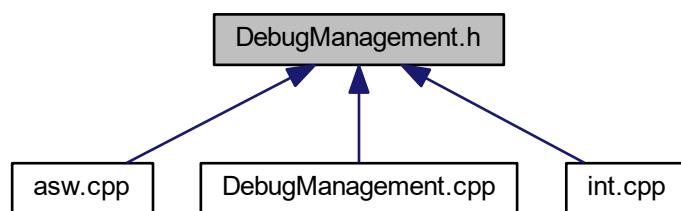
[Watchdog](#) timeout update selection.

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

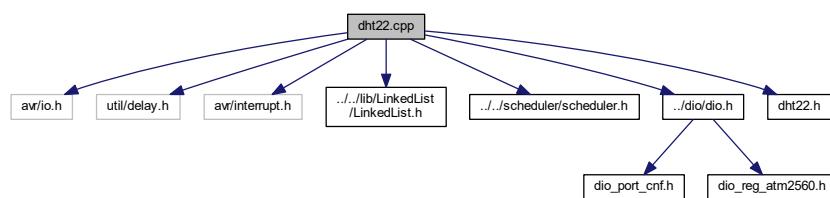
Definition at line 33 of file `DebugManagement.cpp`.

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
```

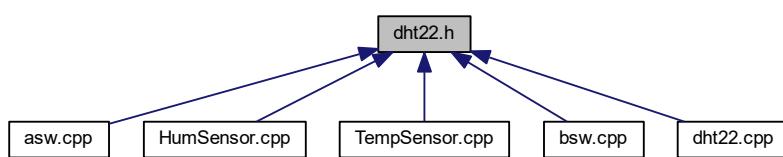
Pointer to [dht22](#) driver object

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`.

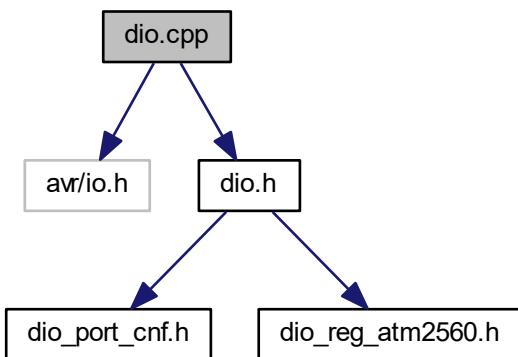
5.15 dio.cpp File Reference

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

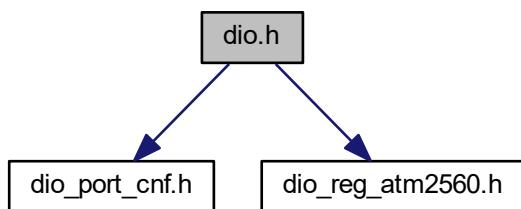
Definition at line 14 of file dio.cpp.

5.16 dio.h File Reference

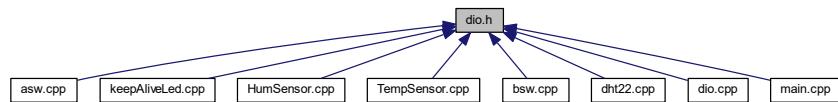
DIO library header file.

```
#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`

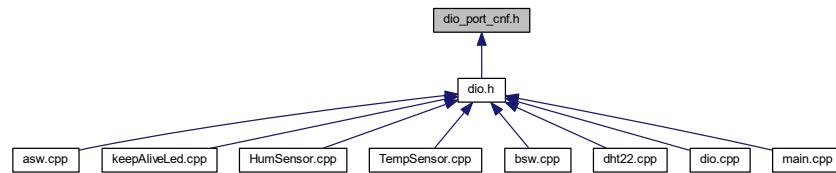
Pointer to dio driver object

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_DDRB

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

Defines the configuration of DDRB register.

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

PB0 : N/A
 PB1 : N/A
 PB2 : N/A
 PB3 : N/A
 PB4 : 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.

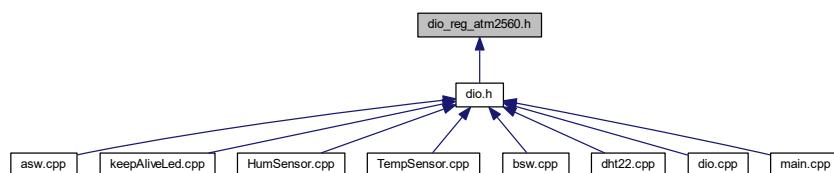
PB0 : N/A
 PB1 : N/A
 PB2 : N/A
 PB3 : N/A
 PB4 : Pull-up
 PB5 : N/A
 PB6 : HIGH
 PB7 : LOW

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

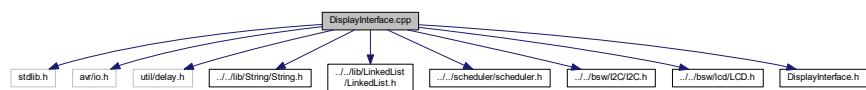
Definition at line 17 of file dio_reg_atm2560.h.

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`

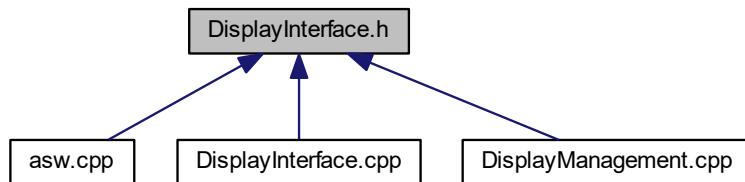
Pointer to `DisplayInterface` object

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

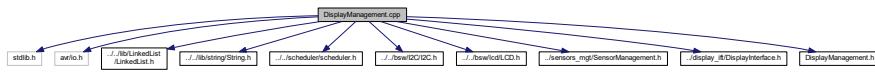
Definition at line 25 of file DisplayInterface.cpp.

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 !"`

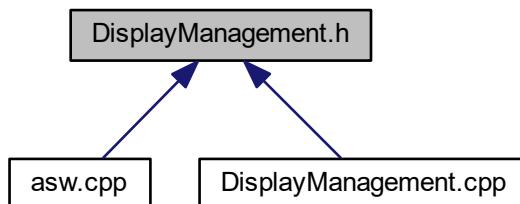
`String` displayed on the screen at startup

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

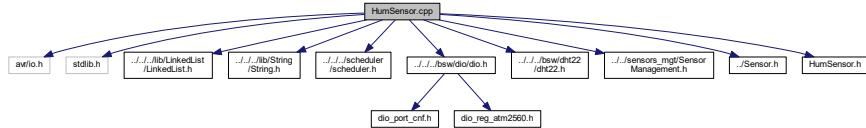
Definition at line 25 of file DisplayManagement.cpp.

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)
```

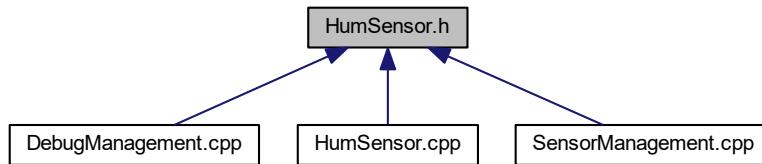
DHT22 is connected to port PB6

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

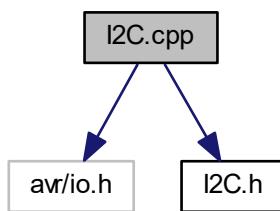
5.25 I2C.cpp File Reference

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`

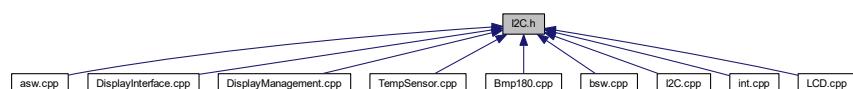
Pointer to [I2C](#) driver object

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.

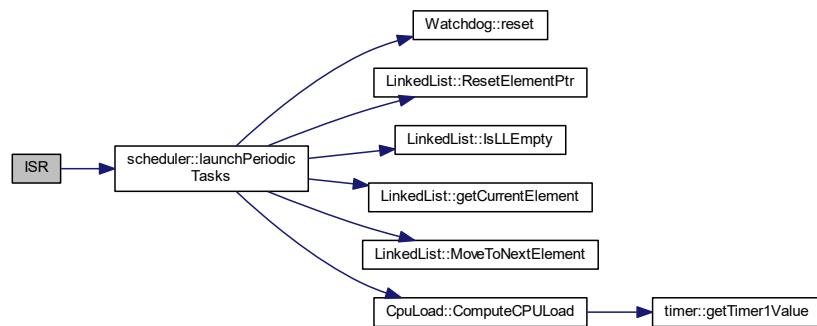
This function handles the interrupt raised by Timer #1. It wakes up the software every 500 ms to perform applications.

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.

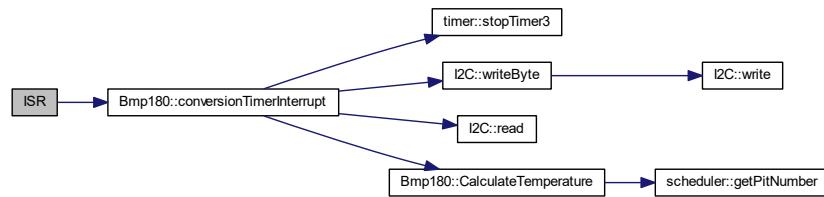
This function calls the end of conversion function of BMP180 driver.

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.

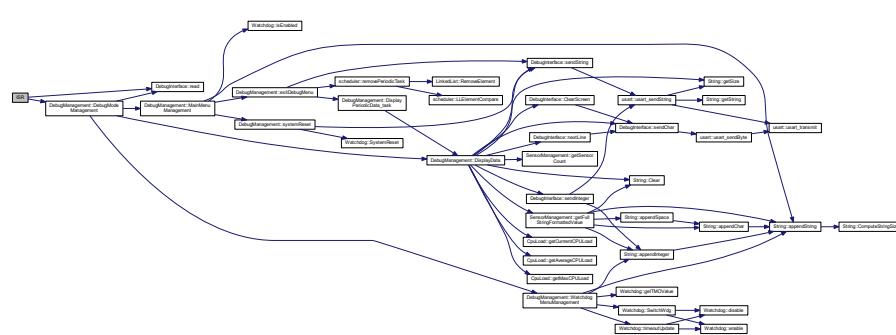
This function handles the interrupt raised when a frame has been received by USART. If debug mode mode is active, it calls debug mode management function. If inactive, it calls debug mode activation function if the received character is 'a'

Returns

Nothing

Definition at line 58 of file int.cpp.

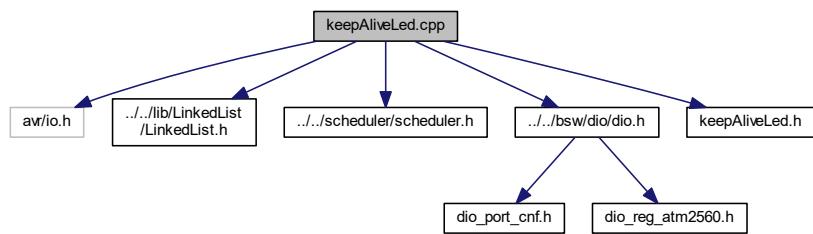
Here is the call graph for this function:



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`

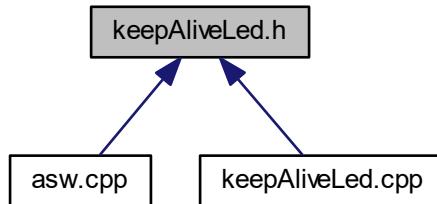
Pointer to [keepAliveLed](#) object

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

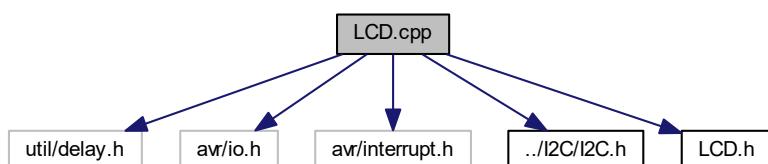
Definition at line 20 of file keepAliveLed.cpp.

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`

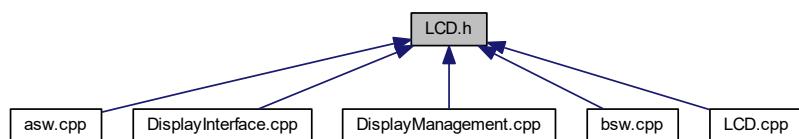
Pointer to `LCD` driver object

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_DATADDRAM	
LCD_DATACGRAM	

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

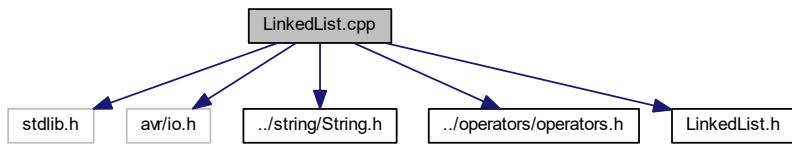
Definition at line 17 of file `LCD.cpp`.

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

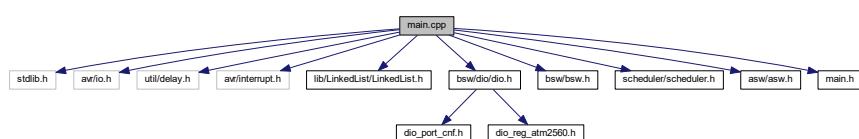
Definition at line 14 of file `LinkedList.h`.

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()

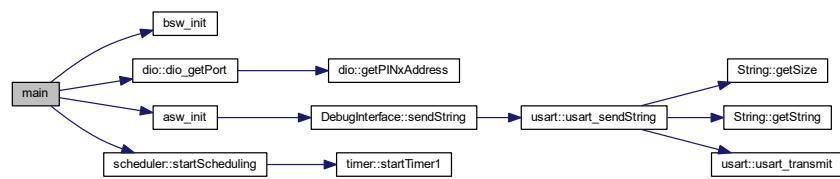
```
int main (
    void )
```

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
```

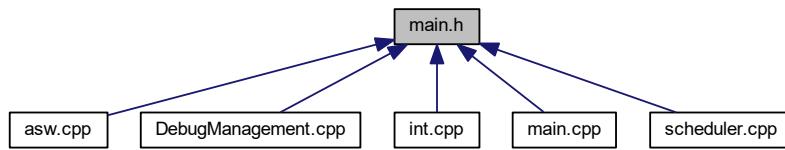
Flag indicating if the debug mode is activated or not

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

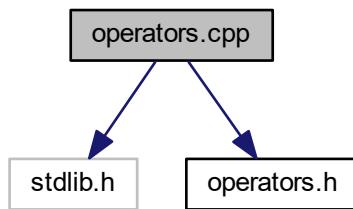
Definition at line 28 of file main.cpp.

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

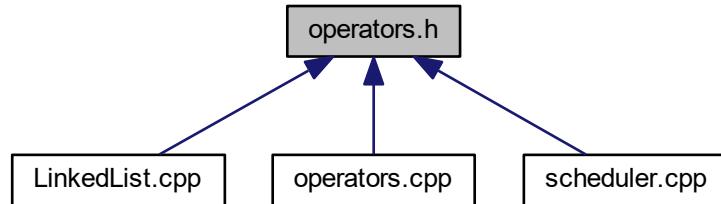
Pointer to the start of allocated memory zone

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

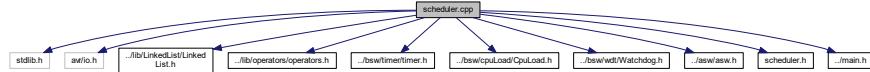
Definition at line 13 of file operators.cpp.

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`

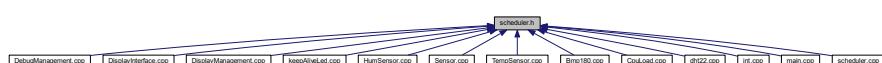
Pointer to scheduler object

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
```

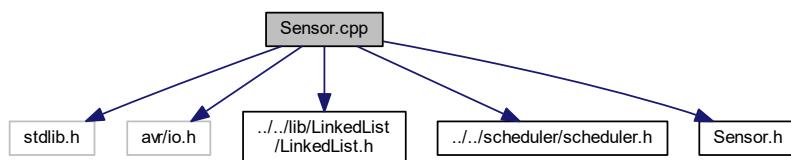
Pointer to scheduler object

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
```

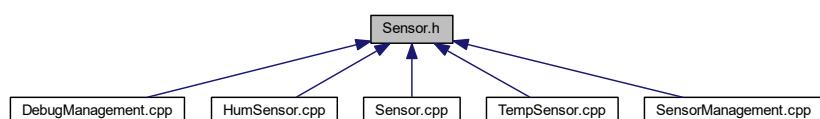
[Sensor](#) data are declared invalid after 30s

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

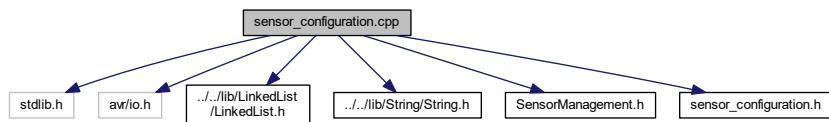
Author

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*) "%"
    }
}
```

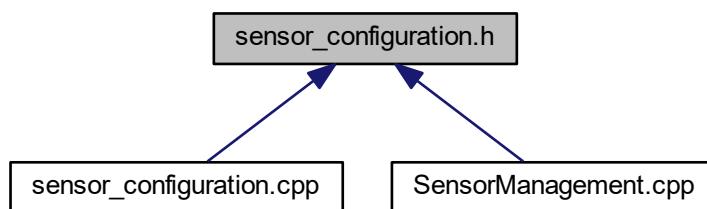
`Sensor` configuration table.

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.

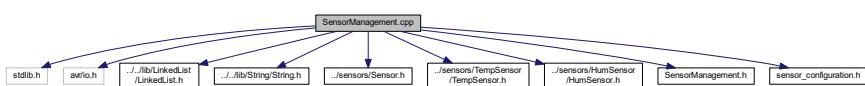
Definition at line 25 of file `sensor_configuration.cpp`.

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

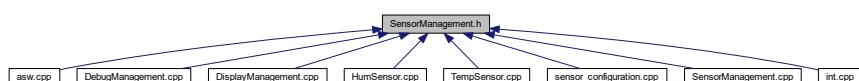
Pointer to the [SensorManagement](#) object

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`

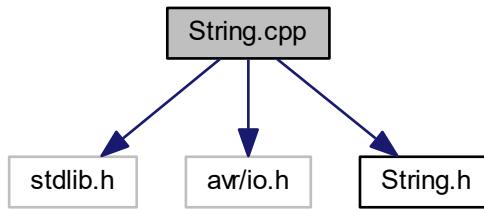
Pointer to the `SensorManagement` object

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

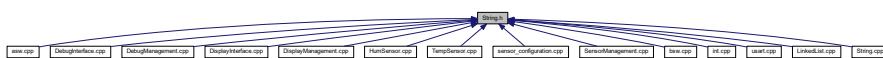
Author

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

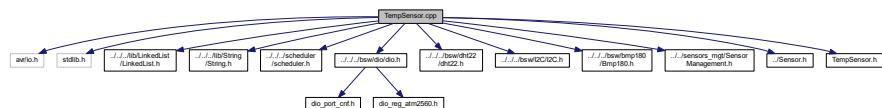
nicls67

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/dio/dio.h"
#include "../../../../bsw/dht22/dht22.h"
#include "../../../../bsw/I2C/I2C.h"
#include "../../../../bsw/bmp180/Bmp180.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 26 of file TempSensor.cpp.

5.48.2.2 TEMP_SENSOR_VALUE_ACCEPTABLE_DIFFERENCE

```
#define TEMP_SENSOR_VALUE_ACCEPTABLE_DIFFERENCE 10
```

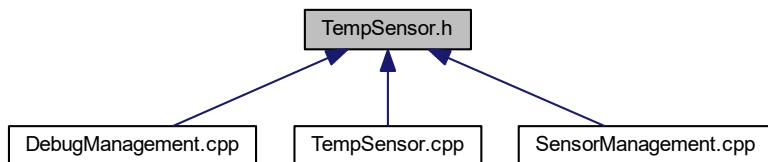
Sensors value can have a difference of 1 degree before becoming invalid

Definition at line 27 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

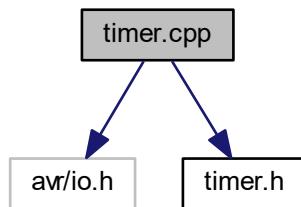
Author

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`

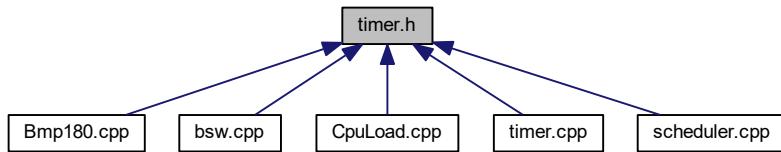
Pointer to timer driver object

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

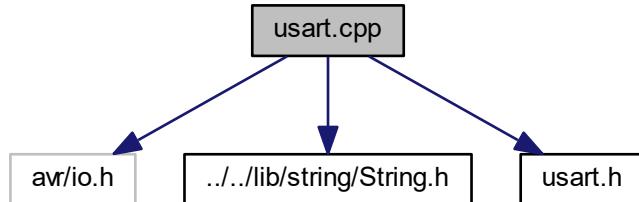
Definition at line 13 of file timer.cpp.

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`

`uart*` `p_global_BSW_usart`

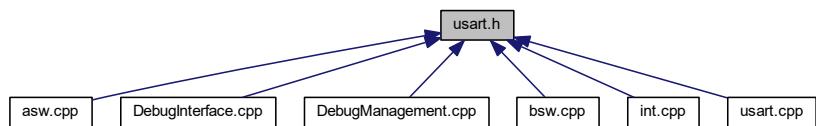
Pointer to usart driver object

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 `uart`
USART serial bus class.

Variables

- `uart * 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
```

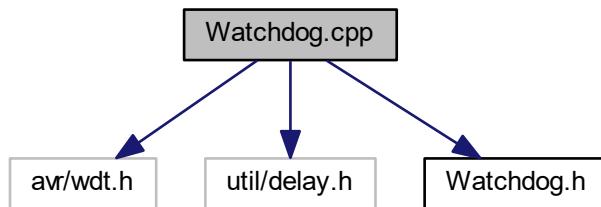
Pointer to usart driver object

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
```

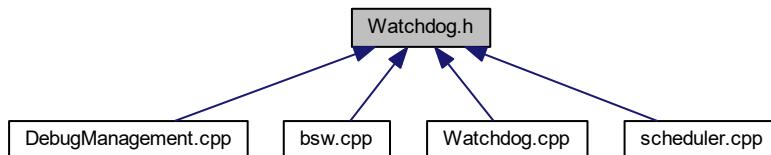
Pointer to [Watchdog](#) driver object

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, 108

~String
 String, 138

AC1
 Bmp180::T_BMP180_calib_data, 148

AC2
 Bmp180::T_BMP180_calib_data, 148

AC3
 Bmp180::T_BMP180_calib_data, 148

AC4
 Bmp180::T_BMP180_calib_data, 148

AC5
 Bmp180::T_BMP180_calib_data, 148

AC6
 Bmp180::T_BMP180_calib_data, 148

ASW_init_cnf
 main.cpp, 252
 main.h, 253

addPeriodicTask
 scheduler, 115

alignment
 T_display_data, 151

appendBool
 String, 139

appendChar
 String, 139

appendInteger
 String, 140

appendSpace
 String, 141

appendString
 String, 142

asw.cpp, 183
 asw_init, 184

asw.h, 185
 asw_init, 185

asw_init
 asw.cpp, 184
 asw.h, 185

AttachNewElement
 LinkedList, 109

avg_load
 CpuLoad, 21

B1
 Bmp180::T_BMP180_calib_data, 149

B2
 Bmp180::T_BMP180_calib_data, 149

BACKLIGHT_PIN
 LCD.h, 239

BMP180_CHIP_ID_CALIB_EEP_START_ADDR
 Bmp180.h, 189

BMP180_CHIP_ID_EEP_ADDR
 Bmp180.h, 189

BMP180_CHIP_ID_EXPECTED
 Bmp180.h, 189

BMP180_CTRL_MEAS_EEP_ADDR
 Bmp180.h, 189

BMP180_CTRL_MEAS_START_TEMP_CONV
 Bmp180.h, 190

BMP180_DATA_VALIDITY_TMO
 Bmp180.h, 190

BMP180_I2C_ADDR
 Bmp180.h, 190

BMP180_I2C_BITRATE
 Bmp180.h, 190

BMP180_OUT_REG_LSB EEPROM_ADDR
 Bmp180.h, 190

BMP180_OUT_REG_MSB EEPROM_ADDR
 Bmp180.h, 191

BMP180_TEMP_MEAS_TIMER_CTC_VALUE
 Bmp180.h, 191

BMP180_TIMER_PRESCALER_VALUE
 Bmp180.h, 191

backlight_en
 T_LCD_conf_struct, 155

backlight_enable
 LCD, 104

BaudRate
 uart, 174

bitrate
 I2C, 87

blinkLed_task
 keepAliveLed, 88

Bmp180, 9
 Bmp180, 11
 CalculateTemperature, 12
 calibration_data, 17
 chip_id, 17
 conversionInProgress, 17
 conversionTimerInterrupt, 12
 getTemperatureValue, 13
 i2c_drv_ptr, 17
 OK, 11
 pressure_value, 17
 readCalibData, 14
 readChipID, 15

startNewTemperatureConversion, 16
 status, 18
 T_BMP180_conversion_type, 10
 T_BMP180_status, 11
 temperature_value, 18
 Bmp180.cpp, 186
 p_global_BSW_bmp180, 187
 Bmp180.h, 188
 BMP180_CHIP_ID_CALIB_EEP_START_ADDR, 189
 BMP180_CHIP_ID_EEP_ADDR, 189
 BMP180_CHIP_ID_EXPECTED, 189
 BMP180_CTRL_MEAS_EEP_ADDR, 189
 BMP180_CTRL_MEAS_START_TEMP_CONV, 190
 BMP180_DATA_VALIDITY_TMO, 190
 BMP180_I2C_ADDR, 190
 BMP180_I2C_BITRATE, 190
 BMP180_OUT_REG_LSB EEPROM_ADDR, 190
 BMP180_OUT_REG_MSB EEPROM_ADDR, 191
 BMP180_TEMP_MEAS_TIMER_CTC_VALUE, 191
 BMP180_TIMER_PRESCALER_VALUE, 191
 p_global_BSW_bmp180, 191
 Bmp180::T_BMP180_calib_data, 147
 AC1, 148
 AC2, 148
 AC3, 148
 AC4, 148
 AC5, 148
 AC6, 148
 B1, 149
 B2, 149
 MB, 149
 MC, 149
 MD, 149
 Bmp180::T_BMP180_measurement_data, 150
 ready, 150
 ts, 150
 value, 150
 bsw.cpp, 192
 bsw_init, 192
 bsw.h, 193
 bsw_init, 194
 bsw_init
 bsw.cpp, 192
 bsw.h, 194
 CalculateTemperature
 Bmp180, 12
 calibration_data
 Bmp180, 17
 chip_id
 Bmp180, 17
 Clear
 String, 143
 ClearFullScreen
 DisplayInterface, 62
 ClearLine

DisplayInterface, 63
 ClearScreen
 DebugInterface, 26
 ClearStringInDataStruct
 DisplayInterface, 64
 cnfCursorBlink
 LCD, 104
 cnfCursorOnOff
 LCD, 104
 cnfDisplayOnOff
 LCD, 104
 cnfEntryModeDir
 LCD, 104
 cnfEntryModeShift
 LCD, 104
 cnfFontType
 LCD, 105
 cnfI2C_addr
 LCD, 105
 cnfLineNumber
 LCD, 105
 command
 LCD, 92
 CompareFctPtr_t
 LinkedList.h, 250
 ComputeCPUload
 CpuLoad, 19
 ComputeStringSize
 String, 143
 ConfigureBacklight
 LCD, 93
 ConfigureCursorBlink
 LCD, 93
 ConfigureCursorOnOff
 LCD, 94
 ConfigureDisplayOnOff
 LCD, 95
 ConfigureEntryModeDir
 LCD, 95
 ConfigureEntryModeShift
 LCD, 96
 ConfigureFontType
 LCD, 97
 ConfigureI2CAddr
 LCD, 97
 ConfigureLineNumber
 LCD, 98
 configureTimer1
 timer, 165
 configureTimer3
 timer, 166
 conversionInProgress
 Bmp180, 17
 conversionTimerInterrupt
 Bmp180, 12
 CpuLoad, 18
 avg_load, 21
 ComputeCPUload, 19

CpuLoad, 19
current_load, 22
getAverageCPULoad, 20
getCurrentCPULoad, 20
getMaxCPULoad, 21
last_sum_value, 22
max_load, 22
sample_cnt, 22
sample_idx, 22
sample_mem, 23
CpuLoad.cpp, 195
 p_global_BSW_cpuload, 195
CpuLoad.h, 196
 NB_OF_SAMPLES, 196
 p_global_BSW_cpuload, 197
curElement_ptr
 LinkedList, 113
current_load
 CpuLoad, 22
cursor_en
 T_LCD_conf_struct, 155
cursorBlink_en
 T_LCD_conf_struct, 155

DATA_ACK
 I2C.h, 229
DDRA_PTR
 dio_reg_atm2560.h, 215
DDRB_PTR
 dio_reg_atm2560.h, 215
DDRC_PTR
 dio_reg_atm2560.h, 215
DDRD_PTR
 dio_reg_atm2560.h, 216
DEBUG_ACTIVE_PORT
 main.cpp, 251
DECODE_PIN
 dio.h, 210
DECODE_PORT
 dio.h, 211
DHT22_PORT
 HumSensor.cpp, 226
 TempSensor.cpp, 270
DISPLAY_LINE_SHIFT_PERIOD_MS
 DisplayInterface.h, 220
DISPLAY_LINE_SHIFT_TEMPO_TIME
 DisplayInterface.h, 220
DISPLAY_MGT_FIRST_LINE_SENSORS
 DisplayManagement.h, 224
DISPLAY_MGT_I2C_BITRATE
 DisplayManagement.h, 224
DISPLAY_MGT_LCD_I2C_ADDR
 DisplayManagement.h, 224
DISPLAY_MGT_PERIOD_TASK_SENSOR
 DisplayManagement.h, 224
DISPLAY_MGT_PERIOD_WELCOME_MSG_REMOTE
 VAL
 DisplayManagement.h, 225
data_name_str
 T_SensorManagement_Sensor_Config, 158
data_ptr
 LinkedList::T_LL_element, 157
ddram_addr
 LCD, 105
debug_ift_ptr
 DebugManagement, 42
debug_mgt_main_menu_state_t
 DebugManagement.h, 205
debug_mgt_state_struct_t, 23
 main_state, 23
 wdg_state, 24
debug_mgt_wdg_state_t
 DebugManagement.h, 205
debug_state
 DebugManagement, 42
DebugInterface, 24
 ClearScreen, 26
 DebugInterface, 25
 nextLine, 26
 read, 27
 sendBool, 27
 sendChar, 28
 sendInteger, 29
 sendString, 30, 31
 usart_drv_ptr, 31
DebugInterface.cpp, 197
 p_global_ASW_DebugInterface, 198
DebugInterface.h, 198
 p_global_ASW_DebugInterface, 199
 USART_BAUDRATE, 199
DebugManagement, 32
 debug_ift_ptr, 42
 debug_state, 42
 DebugManagement, 33
 DebugModeManagement, 34
 DisplayData, 35
 DisplayPeriodicData_task, 36
 exitDebugMenu, 37
 getIfPtr, 38
 getInfoStringPtr, 38
 getMenuStringPtr, 38
 info_string_ptr, 42
 isInfoStringDisplayed, 43
 MainMenuManagement, 39
 menu_string_ptr, 43
 sensorMgt_ptr, 43
 setInfoStringPtr, 40
 systemReset, 40
 WatchdogMenuManagement, 41
DebugManagement.cpp, 200
 p_global_ASW_DebugManagement, 201
 str_debug_info_message_wdg_disabled, 201
 str_debug_info_message_wdg_enabled, 201
 str_debug_info_message_wdg_tmo_updated, 201
 str_debug_info_message_wdg_tmo_value, 202
 str_debug_info_message_wrong_menu_selection, 202

str_debug_main_menu, 202
 str_debug_wdg_menu, 202
 str_debug_wdg_timeout_update_selection, 203
 DebugManagement.h, 203
 debug_mgt_main_menu_state_t, 205
 debug_mgt_wdg_state_t, 205
 p_global_ASW_DebugManagement, 205
 PERIOD_MS_TASK_DISPLAY_CPU_LOAD, 204
 PERIOD_MS_TASK_DISPLAY_DEBUG_DATA,
 204
 DebugModeManagement
 DebugManagement, 34
 dht22, 44
 dht22, 45
 dht22_port, 50
 dio_ptr, 50
 getHumidity, 45
 getTemperature, 47
 initializeCommunication, 48
 mem_humidity, 50
 mem_temperature, 50
 mem_validity, 50
 pit_last_read, 50
 read, 49
 dht22.cpp, 206
 MAX_WAIT_TIME_US, 207
 p_global_BSW_dht22, 207
 dht22.h, 207
 p_global_BSW_dht22, 208
 dht22_port
 dht22, 50
 dio, 51
 dio, 52
 dio_changePortPinCnf, 52
 dio_getPort, 53
 dio_getPort_fast, 54
 dio_invertPort, 54
 dio_memorizePINaddress, 55
 dio_setPort, 56
 getDDRxAddress, 57
 getPINxAddress, 58
 getPORTxAddress, 58
 PINx_addr_mem, 59
 PINx_idx_mem, 60
 ports_init, 59
 dio.cpp, 208
 p_global_BSW_dio, 209
 dio.h, 209
 DECODE_PIN, 210
 DECODE_PORT, 211
 ENCODE_PORT, 211
 p_global_BSW_dio, 212
 PORT_CNF_IN, 211
 PORT_CNF_OUT, 211
 dio_changePortPinCnf
 dio, 52
 dio_getPort
 dio, 53
 dio_getPort_fast
 dio, 54
 dio_invertPort
 dio, 54
 dio_memorizePINaddress
 dio, 55
 dio_port_cnf.h, 212
 PORT_A, 213
 PORT_B, 213
 PORT_C, 213
 PORT_D, 213
 PORTB_CNF_DDRB, 213
 PORTB_CNF_PORTB, 214
 dio_ptr
 dht22, 50
 dio_reg_atm2560.h, 214
 DDRA_PTR, 215
 DDRB_PTR, 215
 DDRC_PTR, 215
 DDRD_PTR, 216
 PINA_PTR, 216
 PINB_PTR, 216
 PINC_PTR, 216
 PIND_PTR, 216
 PORTA_PTR, 217
 PORTB_PTR, 217
 PORTC_PTR, 217
 PORTD_PTR, 217
 dio_setPort
 dio, 56
 disable
 Watchdog, 177
 display_data
 DisplayInterface, 72
 display_en
 T_LCD_conf_struct, 155
 display_str
 T_display_data, 152
 DisplayData
 DebugManagement, 35
 DisplayFullLine
 DisplayInterface, 65, 66
 DisplayInterface, 60
 ClearFullScreen, 62
 ClearLine, 63
 ClearStringInDataStruct, 64
 display_data, 72
 DisplayFullLine, 65, 66
 DisplayInterface, 62
 dummy, 72
 FindFirstCharAddr, 66
 getDisplayDataPtr, 67
 IsLineEmpty, 68
 isShiftInProgress, 72
 p_lcd, 72
 RefreshLine, 68
 setLineAlignment, 69
 setLineAlignmentAndRefresh, 70

shiftLine_task, 70
updateLineAndRefresh, 71
DisplayInterface.cpp, 218
 p_global_ASW_DisplayInterface, 218
DisplayInterface.h, 219
 DISPLAY_LINE_SHIFT_PERIOD_MS, 220
 DISPLAY_LINE_SHIFT_TEMPO_TIME, 220
 p_global_ASW_DisplayInterface, 221
 T_DisplayInterface_LineAlignment, 220
 T_DisplayInterface_LineDisplayMode, 221
DisplayManagement, 73
 DisplayManagement, 74
 DisplaySensorData_Task, 75
 GetIftPointer, 75
 GetSensorMgtPtr, 76
 p_SensorMgt, 77
 p_display_ift, 77
 RemoveWelcomeMessage_Task, 76
DisplayManagement.cpp, 222
 noSensorsDisplayString, 222
 p_global_ASW_DisplayManagement, 222
 welcomeMessageString, 223
DisplayManagement.h, 223
 DISPLAY_MGT_FIRST_LINE_SENSORS, 224
 DISPLAY_MGT_I2C_BITRATE, 224
 DISPLAY_MGT_LCD_I2C_ADDR, 224
 DISPLAY_MGT_PERIOD_TASK_SENSOR, 224
 DISPLAY_MGT_PERIOD_WELCOME_MSG_R←
 EMOVAL, 225
 LCD_init_cnf, 225
 p_global_ASW_DisplayManagement, 225
DisplayPeriodicData_task
 DebugManagement, 36
DisplaySensorData_Task
 DisplayManagement, 75
dummy
 DisplayInterface, 72

EN_PIN
 LCD.h, 239
ENCODE_PORT
 dio.h, 211
enable
 Watchdog, 177
entryModeDir
 T_LCD_conf_struct, 155
entryModeShift
 T_LCD_conf_struct, 155
exitDebugMenu
 DebugManagement, 37

FindElement
 LinkedList, 109
FindFirstCharAddr
 DisplayInterface, 66
firstElement
 LinkedList, 113
fontType_cnf
 T_LCD_conf_struct, 156

getAverageCPUload
 CpuLoad, 20
getCurrentElement
 LinkedList, 110
getcurrCPUload
 CpuLoad, 20
GetDDRAMAddress
 LCD, 99
getDDRxAddress
 dio, 57
getDisplayDataPtr
 DisplayInterface, 67
getFullStringFormattedValue
 SensorManagement, 132
getHumidity
 dht22, 45
GetIftPointer
 DisplayManagement, 75
getIftPtr
 DebugManagement, 38
getInfoStringPtr
 DebugManagement, 38
GetLineNumberCnf
 LCD, 99
getMaxCPUload
 CpuLoad, 21
getMenuStringPtr
 DebugManagement, 38
getPINxAddress
 dio, 58
getPORTxAddress
 dio, 58
getPitNumber
 scheduler, 116
getRawDataPtr
 Sensor, 125
getSensorCount
 SensorManagement, 133
GetSensorMgtPtr
 DisplayManagement, 76
getSensorObjectPtr
 SensorManagement, 133
getSize
 String, 144
getString
 String, 144
getTMOValue
 Watchdog, 178
getTaskCount
 scheduler, 117
getTaskPeriod
 Sensor, 125
getTemperature
 dht22, 47
getTemperatureValue
 Bmp180, 13
getTimer1Value
 timer, 166

getValidity
 Sensor, 126

getValue
 Sensor, 126

getValueDecimal
 Sensor, 126

getValueInteger
 Sensor, 127

HumSensor, 78
 HumSensor, 79
 readHumSensor_task, 80
 updateTaskPeriod, 81

HumSensor.cpp, 226
 DHT22_PORT, 226

HumSensor.h, 227

I2C.cpp, 227
 p_global_BSW_i2c, 228

I2C.h, 228
 DATA_ACK, 229
 p_global_BSW_i2c, 230
 REPEATED_START, 229
 SLAR_ACK, 229
 SLAW_ACK, 230
 START, 230

I2C, 82
 bitrate, 87
 I2C, 83
 initializeBus, 83
 read, 84
 setBitRate, 85
 write, 85
 writeByte, 86

i2c_addr
 T_LCD_conf_struct, 156

i2c_bitrate
 T_LCD_conf_struct, 156

i2c_drv_ptr
 Bmp180, 17
 LCD, 105

ISR
 int.cpp, 231–233

info_string_ptr
 DebugManagement, 42

initializeBus
 I2C, 83

initializeCommunication
 dht22, 48

InitializeScreen
 LCD, 99

int.cpp, 231
 ISR, 231–233

isActive
 Watchdog, 182

isDebugEnabled
 T_ASW_init_cnf, 146

isDebugModeActivated
 main.cpp, 252

main.h, 253

isDisplayActivated
 T_ASW_init_cnf, 146

isEmpty
 T_display_data, 152

isEnabled
 Watchdog, 178

isInfoStringDisplayed
 DebugManagement, 43

isLEDActivated
 T_ASW_init_cnf, 146

IsLLEmpty
 LinkedList, 110

IsLineEmpty
 DisplayInterface, 68

isSensorMgtActivated
 T_ASW_init_cnf, 147

isShiftInProgress
 DisplayInterface, 72

keepAliveLed, 87
 blinkLed_task, 88
 keepAliveLed, 88

keepAliveLed.cpp, 234
 p_global_ASW_keepAliveLed, 234

keepAliveLed.h, 235
 LED_PORT, 235
 p_global_ASW_keepAliveLed, 236
 PERIOD_MS_TASK_LED, 236

LCD.cpp, 236
 p_global_BSW_lcd, 237

LCD.h, 237
 BACKLIGHT_PIN, 239
 EN_PIN, 239
 LCD_CNF_BACKLIGHT_OFF, 239
 LCD_CNF_BACKLIGHT_ON, 239
 LCD_CNF_CURSOR_BLINK_OFF, 240
 LCD_CNF_CURSOR_BLINK_ON, 240
 LCD_CNF_CURSOR_OFF, 240
 LCD_CNF_CURSOR_ON, 240
 LCD_CNF_DISPLAY_OFF, 240
 LCD_CNF_DISPLAY_ON, 241
 LCD_CNF_ENTRY_MODE_DIRECTION_LEFT,
 241
 LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT,
 241
 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT←
 OFF, 241
 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT←
 ON, 241
 LCD_CNF_FONT_5_11, 242
 LCD_CNF_FONT_5_8, 242
 LCD_CNF_ONE_LINE, 242
 LCD_CNF_SHIFT_ID, 242
 LCD_CNF_SHIFT_SH, 242
 LCD_CNF_TWO_LINE, 243
 LCD_DISPLAY_CTRL_FIELD_B, 243
 LCD_DISPLAY_CTRL_FIELD_C, 243

LCD_DISPLAY_CTRL_FIELD_D, 243
LCD_FCT_SET_FIELD_DL, 243
LCD_FCT_SET_FIELD_F, 244
LCD_FCT_SET_FIELD_N, 244
LCD_INST_CLR_DISPLAY_BIT, 244
LCD_INST_DISPLAY_CTRL, 244
LCD_INST_ENTRY_MODE_SET, 244
LCD_INST_FUNCTION_SET, 245
LCD_INST_SET_DDRAM_ADDR, 245
LCD_RAM_1_LINE_MAX, 245
LCD_RAM_1_LINE_MIN, 245
LCD_RAM_2_LINES_MAX_1, 245
LCD_RAM_2_LINES_MAX_2, 246
LCD_RAM_2_LINES_MIN_1, 246
LCD_RAM_2_LINES_MIN_2, 246
LCD_SIZE_NB_CHAR_PER_LINE, 246
LCD_SIZE_NB_LINES, 246
LCD_WAIT_CLR_RETURN, 247
LCD_WAIT_OTHER_MODES, 247
p_global_BSW_lcd, 248
RS_PIN, 247
RW_PIN, 247
T_LCD_command, 247
T_LCD_config_mode, 248
T_LCD_ram_area, 248
LCD_CNF_BACKLIGHT_OFF
 LCD.h, 239
LCD_CNF_BACKLIGHT_ON
 LCD.h, 239
LCD_CNF_CURSOR_BLINK_OFF
 LCD.h, 240
LCD_CNF_CURSOR_BLINK_ON
 LCD.h, 240
LCD_CNF_CURSOR_OFF
 LCD.h, 240
LCD_CNF_CURSOR_ON
 LCD.h, 240
LCD_CNF_DISPLAY_OFF
 LCD.h, 240
LCD_CNF_DISPLAY_ON
 LCD.h, 241
LCD_CNF_ENTRY_MODE_DIRECTION_LEFT
 LCD.h, 241
LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT
 LCD.h, 241
LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF
 LCD.h, 241
LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON
 LCD.h, 241
LCD_CNF_FONT_5_11
 LCD.h, 242
LCD_CNF_FONT_5_8
 LCD.h, 242
LCD_CNF_ONE_LINE
 LCD.h, 242
LCD_CNF_SHIFT_ID
 LCD.h, 242
LCD_CNF_SHIFT_SH

LCD.h, 242
LCD_CNF_TWO_LINE
 LCD.h, 243
LCD_DISPLAY_CTRL_FIELD_B
 LCD.h, 243
LCD_DISPLAY_CTRL_FIELD_C
 LCD.h, 243
LCD_DISPLAY_CTRL_FIELD_D
 LCD.h, 243
LCD_FCT_SET_FIELD_DL
 LCD.h, 243
LCD_FCT_SET_FIELD_F
 LCD.h, 244
LCD_FCT_SET_FIELD_N
 LCD.h, 244
LCD_INST_CLR_DISPLAY_BIT
 LCD.h, 244
LCD_INST_DISPLAY_CTRL
 LCD.h, 244
LCD_INST_ENTRY_MODE_SET
 LCD.h, 244
LCD_INST_FUNCTION_SET
 LCD.h, 245
LCD_INST_SET_DDRAM_ADDR
 LCD.h, 245
LCD_RAM_1_LINE_MAX
 LCD.h, 245
LCD_RAM_1_LINE_MIN
 LCD.h, 245
LCD_RAM_2_LINES_MAX_1
 LCD.h, 245
LCD_RAM_2_LINES_MAX_2
 LCD.h, 246
LCD_RAM_2_LINES_MIN_1
 LCD.h, 246
LCD_RAM_2_LINES_MIN_2
 LCD.h, 246
LCD_SIZE_NB_CHAR_PER_LINE
 LCD.h, 246
LCD_SIZE_NB_LINES
 LCD.h, 246
LCD_WAIT_CLR_RETURN
 LCD.h, 247
LCD_WAIT_OTHER_MODES
 LCD.h, 247
LCD_init_cnf
 DisplayManagement.h, 225
LCD, 89
 backlight_enable, 104
 cnfCursorBlink, 104
 cnfCursorOnOff, 104
 cnfDisplayOnOff, 104
 cnfEntryModeDir, 104
 cnfEntryModeShift, 104
 cnfFontType, 105
 cnfI2C_addr, 105
 cnfLineNumber, 105
 command, 92

ConfigureBacklight, 93
 ConfigureCursorBlink, 93
 ConfigureCursorOnOff, 94
 ConfigureDisplayOnOff, 95
 ConfigureEntryModeDir, 95
 ConfigureEntryModeShift, 96
 ConfigureFontType, 97
 ConfigureI2CAddr, 97
 ConfigureLineNumber, 98
 ddram_addr, 105
 GetDDRAMAddress, 99
 GetLineNumberCnf, 99
 i2c_drv_ptr, 105
 InitializeScreen, 99
 LCD, 91
 SetDDRAMAddress, 100
 write, 101
 write4bits, 102
 WriteInRam, 103
LED_PORT
 keepAliveLed.h, 235
LLElementCompare
 scheduler, 118
last_sum_value
 CpuLoad, 22
launchPeriodicTasks
 scheduler, 117
lineNumber_cnf
 T_LCD_conf_struct, 156
LinkedList, 106
 ~LinkedList, 108
 AttachNewElement, 109
 curElement_ptr, 113
 FindElement, 109
 firstElement, 113
 getCurrentElement, 110
 IsLLEmpty, 110
 LinkedList, 108
 MoveToNextElement, 111
 RemoveElement, 111
 ResetElementPtr, 112
 T_LL_element, 107
LinkedList.cpp, 249
LinkedList.h, 249
 CompareFctPtr_t, 250
LinkedList::T_LL_element, 157
 data_ptr, 157
 nextElement, 157
MAX_WAIT_TIME_US
 dht22.cpp, 207
main
 main.cpp, 251
main.cpp, 250
 ASW_init_cnf, 252
 DEBUG_ACTIVE_PORT, 251
 isDebugModeActivated, 252
 main, 251
main.h, 253
ASW_init_cnf, 253
 isDebugModeActivated, 253
main_state
 debug_mgt_state_struct_t, 23
MainMenuManagement
 DebugManagement, 39
max_load
 CpuLoad, 22
MB
 Bmp180::T_BMP180_calib_data, 149
MC
 Bmp180::T_BMP180_calib_data, 149
MD
 Bmp180::T_BMP180_calib_data, 149
mem_humidity
 dht22, 50
mem_temperature
 dht22, 50
mem_validity
 dht22, 50
menu_string_ptr
 DebugManagement, 43
mode
 T_display_data, 152
MoveToNextElement
 LinkedList, 111
NB_OF_SAMPLES
 CpuLoad.h, 196
nb_sensors
 SensorManagement, 135
nextElement
 LinkedList::T_LL_element, 157
nextLine
 DebugInterface, 26
noSensorsDisplayString
 DisplayManagement.cpp, 222
OK
 Bmp180, 11
operator delete
 operators.cpp, 255
 operators.h, 256
operator new
 operators.cpp, 255
 operators.h, 257
operators.cpp, 254
 operator delete, 255
 operator new, 255
operators.h, 255
 operator delete, 256
 operator new, 257
p_SensorMgt
 DisplayManagement, 77
p_display_ift
 DisplayManagement, 77
p_global_ASW_DebugInterface
 DebugInterface.cpp, 198

DebugInterface.h, 199
p_global_ASW_DebugManagement
 DebugManagement.cpp, 201
 DebugManagement.h, 205
p_global_ASW_DisplayInterface
 DisplayInterface.cpp, 218
 DisplayInterface.h, 221
p_global_ASW_DisplayManagement
 DisplayManagement.cpp, 222
 DisplayManagement.h, 225
p_global_ASW_SensorManagement
 SensorManagement.cpp, 266
 SensorManagement.h, 267
p_global_ASW_keepAliveLed
 keepAliveLed.cpp, 234
 keepAliveLed.h, 236
p_global_BSW_bmp180
 Bmp180.cpp, 187
 Bmp180.h, 191
p_global_BSW_cpupload
 CpuLoad.cpp, 195
 CpuLoad.h, 197
p_global_BSW_dht22
 dht22.cpp, 207
 dht22.h, 208
p_global_BSW_dio
 dio.cpp, 209
 dio.h, 212
p_global_BSW_i2c
 I2C.cpp, 228
 I2C.h, 230
p_global_BSW_lcd
 LCD.cpp, 237
 LCD.h, 248
p_global_BSW_timer
 timer.cpp, 272
 timer.h, 273
p_global_BSW_usart
 USART.cpp, 274
 USART.h, 275
p_global_BSW_wdg
 Watchdog.cpp, 276
 Watchdog.h, 279
p_global_scheduler
 scheduler.cpp, 258
 scheduler.h, 260
p_lcd
 DisplayInterface, 72
PERIOD_MS_TASK_DISPLAY_CPU_LOAD
 DebugManagement.h, 204
PERIOD_MS_TASK_DISPLAY_DEBUG_DATA
 DebugManagement.h, 204
PERIOD_MS_TASK_LED
 keepAliveLed.h, 236
PINA_PTR
 dio_reg_atm2560.h, 216
PINB_PTR
 dio_reg_atm2560.h, 216
PINC_PTR
 dio_reg_atm2560.h, 216
PIND_PTR
 dio_reg_atm2560.h, 216
PINx_addr_mem
 dio, 59
PINx_idx_mem
 dio, 60
PORT_CNF_IN
 dio.h, 211
PORT_CNF_OUT
 dio.h, 211
PORT_A
 dio_port_cnf.h, 213
PORT_B
 dio_port_cnf.h, 213
PORT_C
 dio_port_cnf.h, 213
PORT_D
 dio_port_cnf.h, 213
PORTA_PTR
 dio_reg_atm2560.h, 217
PORTB_CNF_DDRB
 dio_port_cnf.h, 213
PORTB_CNF_PORTB
 dio_port_cnf.h, 214
PORTB_PTR
 dio_reg_atm2560.h, 217
PORTC_PTR
 dio_reg_atm2560.h, 217
PORTD_PTR
 dio_reg_atm2560.h, 217
PRESCALER_PERIODIC_TIMER
 scheduler.h, 259
period
 scheduler::Task_t, 159
 T_SensorManagement_Sensor_Config, 158
pit_last_read
 dht22, 50
pit_number
 scheduler, 122
ports_init
 dio, 59
prescaler1
 timer, 169
prescaler3
 timer, 169
pressure_value
 Bmp180, 17
REPEATED_START
 I2C.h, 229
RS_PIN
 LCD.h, 247
RW_PIN
 LCD.h, 247
raw_data
 Sensor, 130
read

DebugInterface, 27
 dht22, 49
 I2C, 84
 readCalibData
 Bmp180, 14
 readChipID
 Bmp180, 15
 readHumSensor_task
 HumSensor, 80
 readSensor_task
 Sensor, 127
 readTempSensor_task
 TempSensor, 162
 ready
 Bmp180::T_BMP180_measurement_data, 150
 RefreshLine
 DisplayInterface, 68
 RemoveElement
 LinkedList, 111
 removePeriodicTask
 scheduler, 119
 RemoveWelcomeMessage_Task
 DisplayManagement, 76
 reset
 Watchdog, 179
 ResetElementPtr
 LinkedList, 112

 SENSOR_MGT_CNF_DEFAULT_PERIOD
 sensor_configuration.cpp, 263
 SENSOR_MGT_CNF_DEFAULT_TMO
 sensor_configuration.cpp, 263
 SLAR_ACK
 I2C.h, 229
 SLAW_ACK
 I2C.h, 230
 START
 I2C.h, 230
 SW_PERIOD_MS
 scheduler.h, 259
 sample_cnt
 CpuLoad, 22
 sample_idx
 CpuLoad, 22
 sample_mem
 CpuLoad, 23
 scheduler, 113
 addPeriodicTask, 115
 getPitNumber, 116
 getTaskCount, 117
 LLElementCompare, 118
 launchPeriodicTasks, 117
 pit_number, 122
 removePeriodicTask, 119
 scheduler, 115
 startScheduling, 120
 task_count, 122
 Task_t, 115
 TasksLL_ptr, 122

 updateTaskPeriod, 121
 scheduler.cpp, 257
 p_global_scheduler, 258
 scheduler.h, 258
 p_global_scheduler, 260
 PRESCALER_PERIODIC_TIMER, 259
 SW_PERIOD_MS, 259
 TIMER_CTC_VALUE, 259
 TaskPtr_t, 260
 scheduler::Task_t, 159
 period, 159
 TaskPtr, 159
 sendBool
 DebugInterface, 27
 sendChar
 DebugInterface, 28
 sendInteger
 DebugInterface, 29
 sendString
 DebugInterface, 30, 31
 Sensor, 123
 getRawDataPtr, 125
 getTaskPeriod, 125
 getValidity, 126
 getValue, 126
 getValueDecimal, 126
 getValueInteger, 127
 raw_data, 130
 readSensor_task, 127
 Sensor, 124
 setLastValidity, 127
 setValidityTMO, 128
 task_period, 130
 updateTaskPeriod, 128
 updateValidData, 129
 valid_pit, 130
 valid_value, 130
 validity, 130
 validity_last_read, 130
 validity_tmo, 131
 Sensor.cpp, 260
 TASK_PERIOD_DEFAULT, 261
 VALIDITY_TIMEOUT_MS_DEFAULT, 261
 Sensor.h, 261
 sensor_configuration.cpp, 262
 SENSOR_MGT_CNF_DEFAULT_PERIOD, 263
 SENSOR_MGT_CNF_DEFAULT_TMO, 263
 SensorManagement_Sensor_Config_list, 263
 sensor_configuration.h, 264
 SensorManagement_Sensor_Config_list, 265
 sensor_ptr_table
 SensorManagement, 136
 sensor_type
 T_SensorManagement_Sensor_Config, 158
 SensorManagement, 131
 getFullStringFormattedValue, 132
 getSensorCount, 133
 getSensorObjectPtr, 133

nb_sensors, 135
sensor_ptr_table, 136
SensorManagement, 132
updateTaskPeriod, 135
SensorManagement.cpp, 265
 p_global_ASW_SensorManagement, 266
SensorManagement.h, 266
 p_global_ASW_SensorManagement, 267
 T_SensorManagement_Sensor_Type, 267
SensorManagement_Sensor_Config_list
 sensor_configuration.cpp, 263
 sensor_configuration.h, 265
sensorMgt_ptr
 DebugManagement, 43
setBaudRate
 uart, 170
setBitRate
 I2C, 85
SetDDRAMAddress
 LCD, 100
setInfoStringPtr
 DebugManagement, 40
setLastValidity
 Sensor, 127
setLineAlignment
 DisplayInterface, 69
setLineAlignmentAndRefresh
 DisplayInterface, 70
setValidityTMO
 Sensor, 128
shift_data
 T_display_data, 152
shiftLine_task
 DisplayInterface, 70
size
 String, 145
startNewTemperatureConversion
 Bmp180, 16
startScheduling
 scheduler, 120
startTimer1
 timer, 167
startTimer3
 timer, 167
status
 Bmp180, 18
stopTimer1
 timer, 168
stopTimer3
 timer, 168
str_cur_ptr
 T_Display_shift_data, 153
str_debug_info_message_wdg_disabled
 DebugManagement.cpp, 201
str_debug_info_message_wdg_enabled
 DebugManagement.cpp, 201
str_debug_info_message_wdg_tmo_updated
 DebugManagement.cpp, 201
str_debug_info_message_wdg_tmo_value
 DebugManagement.cpp, 202
str_debug_info_message_wrong_menu_selection
 DebugManagement.cpp, 202
str_debug_main_menu
 DebugManagement.cpp, 202
str_debug_wdg_menu
 DebugManagement.cpp, 202
str_debug_wdg_timeout_update_selection
 DebugManagement.cpp, 203
str_ptr
 T_Display_shift_data, 153
String, 136
 ~String, 138
 appendBool, 139
 appendChar, 139
 appendInteger, 140
 appendSpace, 141
 appendString, 142
 Clear, 143
 ComputeStringSize, 143
 getSize, 144
 getString, 144
 size, 145
 String, 137, 138
 string, 145
string
 String, 145
String.cpp, 268
String.h, 268
SwitchWdg
 Watchdog, 179
SystemReset
 Watchdog, 180
systemReset
 DebugManagement, 40
T_ASW_init_cnf, 146
 isDebugEnabled, 146
 isDisplayActivated, 146
 isLEDActivated, 146
 isSensorMgtActivated, 147
T_BMP180_conversion_type
 Bmp180, 10
T_BMP180_status
 Bmp180, 11
T_Display_shift_data, 153
 str_cur_ptr, 153
 str_ptr, 153
 temporization, 154
T_DisplayInterface_LineAlignment
 DisplayInterface.h, 220
T_DisplayInterface_LineDisplayMode
 DisplayInterface.h, 221
T_LCD_Command
 LCD.h, 247
T_LCD_conf_struct, 154
 backlight_en, 155
 cursor_en, 155

cursorBlink_en, 155
 display_en, 155
 entryModeDir, 155
 entryModeShift, 155
 fontType_cnf, 156
 i2c_addr, 156
 i2c_bitrate, 156
 lineNumber_cnf, 156
T_LCD_config_mode
 LCD.h, 248
T_LCD_ram_area
 LCD.h, 248
T_LL_element
 LinkedList, 107
T_SensorManagement_Sensor_Config, 158
 data_name_str, 158
 period, 158
 sensor_type, 158
 unit_str, 158
 validity_tmo, 159
T_SensorManagement_Sensor_Type
 SensorManagement.h, 267
T_display_data, 151
 alignment, 151
 display_str, 152
 isEmpty, 152
 mode, 152
 shift_data, 152
TASK_PERIOD_DEFAULT
 Sensor.cpp, 261
TEMP_SENSOR_VALUE_ACCEPTABLE_DIFFERENCE_NCE
 TempSensor.cpp, 270
TIMER_CTC_VALUE
 scheduler.h, 259
task_count
 scheduler, 122
task_period
 Sensor, 130
Task_t
 scheduler, 115
TaskPtr
 scheduler::Task_t, 159
TaskPtr_t
 scheduler.h, 260
TasksLL_ptr
 scheduler, 122
TempSensor, 160
 readTempSensor_task, 162
TempSensor, 161
 updateTaskPeriod, 163
TempSensor.cpp, 269
 DHT22_PORT, 270
 TEMP_SENSOR_VALUE_ACCEPTABLE_DIFFERENCE, 270
TempSensor.h, 270
temperature_value
 Bmp180, 18
temporization
 T_Display_shift_data, 154
timeoutUpdate
 Watchdog, 181
timer, 164
 configureTimer1, 165
 configureTimer3, 166
 getTimer1Value, 166
 prescaler1, 169
 prescaler3, 169
 startTimer1, 167
 startTimer3, 167
 stopTimer1, 168
 stopTimer3, 168
 timer, 165
timer.cpp, 271
 p_global_BSW_timer, 272
timer.h, 272
 p_global_BSW_timer, 273
tmo_value
 Watchdog, 182
ts
 Bmp180::T_BMP180_measurement_data, 150
USART_BAUDRATE
 DebugInterface.h, 199
unit_str
 T_SensorManagement_Sensor_Config, 158
updateLineAndRefresh
 DisplayInterface, 71
updateTaskPeriod
 HumSensor, 81
 scheduler, 121
 Sensor, 128
 SensorManagement, 135
 TempSensor, 163
updateValidData
 Sensor, 129
uart, 169
 BaudRate, 174
 setBaudRate, 170
 usart, 170
 usart_init, 171
 usart_read, 171
 usart_sendByte, 172
 usart_sendString, 173
 usart_transmit, 173
uart.cpp, 273
 p_global_BSW_usart, 274
uart.h, 274
 p_global_BSW_usart, 275
uart_drv_ptr
 DebugInterface, 31
uart_init
 usart, 171
uart_read
 usart, 171
uart_sendByte
 usart, 172

uart_sendString
 uart, 173
uart_transmit
 uart, 173

VALIDITY_TIMEOUT_MS_DEFAULT
 Sensor.cpp, 261

valid_pit
 Sensor, 130

valid_value
 Sensor, 130

validity
 Sensor, 130

validity_last_read
 Sensor, 130

validity_tmo
 Sensor, 131
 T_SensorManagement_Sensor_Config, 159

value
 Bmp180::T_BMP180_measurement_data, 150

WDG_TIMEOUT_DEFAULT_MS
 Watchdog.cpp, 276

WDG_TMO_120MS
 Watchdog.h, 277

WDG_TMO_15MS
 Watchdog.h, 277

WDG_TMO_1S
 Watchdog.h, 278

WDG_TMO_250MS
 Watchdog.h, 278

WDG_TMO_2S
 Watchdog.h, 278

WDG_TMO_30MS
 Watchdog.h, 278

WDG_TMO_4S
 Watchdog.h, 278

WDG_TMO_500MS
 Watchdog.h, 279

WDG_TMO_60MS
 Watchdog.h, 279

WDG_TMO_8S
 Watchdog.h, 279

Watchdog, 174
 disable, 177
 enable, 177
 getTMOValue, 178
 isActive, 182
 isEnabled, 178
 reset, 179
 SwitchWdg, 179
 SystemReset, 180
 timeoutUpdate, 181
 tmo_value, 182
 Watchdog, 175, 176

Watchdog.cpp, 275
 p_global_BSW_wdg, 276
 WDG_TIMEOUT_DEFAULT_MS, 276

Watchdog.h, 276