

Arduino

1.0

Generated by Doxygen 1.8.14



# Contents

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

4.1.3.11	isPressConversionActivated()	19
4.1.3.12	isTempConversionActivated()	19
4.1.3.13	PressureMonitoring()	20
4.1.3.14	readCalibData()	21
4.1.3.15	readChipID()	21
4.1.3.16	startNewPressureConversion()	22
4.1.3.17	startNewTemperatureConversion()	23
4.1.3.18	StopPressureConversion()	24
4.1.3.19	StopTemperatureConversion()	25
4.1.3.20	TemperatureMonitoring()	25
4.1.4	Member Data Documentation	26
4.1.4.1	B5_mem	26
4.1.4.2	calibration_data	26
4.1.4.3	chip_id	26
4.1.4.4	i2c_drv_ptr	26
4.1.4.5	isPressConvActivated	26
4.1.4.6	isTempConvActivated	27
4.1.4.7	pressure_value	27
4.1.4.8	status	27
4.1.4.9	task_period	27
4.1.4.10	temperature_value	27
4.2	CpuLoad Class Reference	28
4.2.1	Detailed Description	28
4.2.2	Constructor & Destructor Documentation	28
4.2.2.1	CpuLoad()	29
4.2.3	Member Function Documentation	29
4.2.3.1	ComputeCPULoad()	29
4.2.3.2	getAverageCPULoad()	30
4.2.3.3	getCurrentCPULoad()	30
4.2.3.4	getMaxCPULoad()	31

4.2.4	Member Data Documentation . . . . .	31
4.2.4.1	avg_load . . . . .	31
4.2.4.2	current_load . . . . .	31
4.2.4.3	last_sum_value . . . . .	32
4.2.4.4	max_load . . . . .	32
4.2.4.5	sample_cnt . . . . .	32
4.2.4.6	sample_idx . . . . .	32
4.2.4.7	sample_mem . . . . .	32
4.3	debug_mgt_state_struct_t Struct Reference . . . . .	33
4.3.1	Detailed Description . . . . .	33
4.3.2	Member Data Documentation . . . . .	33
4.3.2.1	main_state . . . . .	33
4.3.2.2	wdg_state . . . . .	33
4.4	DebugInterface Class Reference . . . . .	34
4.4.1	Detailed Description . . . . .	35
4.4.2	Constructor & Destructor Documentation . . . . .	35
4.4.2.1	DebugInterface() . . . . .	35
4.4.3	Member Function Documentation . . . . .	35
4.4.3.1	ClearScreen() . . . . .	35
4.4.3.2	nextLine() . . . . .	36
4.4.3.3	read() . . . . .	36
4.4.3.4	sendBool() . . . . .	37
4.4.3.5	sendChar() . . . . .	37
4.4.3.6	sendInteger() . . . . .	38
4.4.3.7	sendString() [1/2] . . . . .	39
4.4.3.8	sendString() [2/2] . . . . .	40
4.4.4	Member Data Documentation . . . . .	40
4.4.4.1	uart_drv_ptr . . . . .	41
4.5	DebugManagement Class Reference . . . . .	41
4.5.1	Detailed Description . . . . .	42

---

4.5.2	Constructor & Destructor Documentation . . . . .	42
4.5.2.1	DebugManagement() . . . . .	43
4.5.3	Member Function Documentation . . . . .	43
4.5.3.1	DebugModeManagement() . . . . .	43
4.5.3.2	DisplayData() . . . . .	44
4.5.3.3	DisplayPeriodicData_task() . . . . .	45
4.5.3.4	exitDebugMenu() . . . . .	46
4.5.3.5	getIiftPtr() . . . . .	47
4.5.3.6	getInfoStringPtr() . . . . .	47
4.5.3.7	getMenuStringPtr() . . . . .	48
4.5.3.8	MainMenuManagement() . . . . .	48
4.5.3.9	setInfoStringPtr() . . . . .	49
4.5.3.10	systemReset() . . . . .	49
4.5.3.11	WatchdogMenuManagement() . . . . .	50
4.5.4	Member Data Documentation . . . . .	51
4.5.4.1	debug_ift_ptr . . . . .	51
4.5.4.2	debug_state . . . . .	51
4.5.4.3	info_string_ptr . . . . .	52
4.5.4.4	isInfoStringDisplayed . . . . .	52
4.5.4.5	menu_string_ptr . . . . .	52
4.5.4.6	sensorMgt_ptr . . . . .	52
4.6	dht22 Class Reference . . . . .	53
4.6.1	Detailed Description . . . . .	54
4.6.2	Constructor & Destructor Documentation . . . . .	54
4.6.2.1	dht22() . . . . .	54
4.6.3	Member Function Documentation . . . . .	54
4.6.3.1	getHumidity() . . . . .	55
4.6.3.2	getTemperature() . . . . .	56
4.6.3.3	initializeCommunication() . . . . .	57
4.6.3.4	read() . . . . .	58

4.6.4	Member Data Documentation . . . . .	59
4.6.4.1	dht22_port . . . . .	59
4.6.4.2	dio_ptr . . . . .	59
4.6.4.3	mem_humidity . . . . .	59
4.6.4.4	mem_temperature . . . . .	59
4.6.4.5	mem_validity . . . . .	59
4.6.4.6	pit_last_read . . . . .	60
4.7	dio Class Reference . . . . .	60
4.7.1	Detailed Description . . . . .	61
4.7.2	Constructor & Destructor Documentation . . . . .	61
4.7.2.1	dio() . . . . .	61
4.7.3	Member Function Documentation . . . . .	61
4.7.3.1	dio_changePortPinCnf() . . . . .	61
4.7.3.2	dio_getPort() . . . . .	62
4.7.3.3	dio_getPort_fast() . . . . .	63
4.7.3.4	dio_invertPort() . . . . .	64
4.7.3.5	dio_memorizePINaddress() . . . . .	64
4.7.3.6	dio_setPort() . . . . .	65
4.7.3.7	getDDRxAddress() . . . . .	66
4.7.3.8	getPINxAddress() . . . . .	67
4.7.3.9	getPORTTxAddress() . . . . .	67
4.7.3.10	ports_init() . . . . .	68
4.7.4	Member Data Documentation . . . . .	68
4.7.4.1	PINx_addr_mem . . . . .	69
4.7.4.2	PINx_idx_mem . . . . .	69
4.8	DisplayInterface Class Reference . . . . .	69
4.8.1	Detailed Description . . . . .	71
4.8.2	Constructor & Destructor Documentation . . . . .	71
4.8.2.1	DisplayInterface() . . . . .	71
4.8.3	Member Function Documentation . . . . .	71

---

4.8.3.1	ClearFullScreen()	72
4.8.3.2	ClearLine()	72
4.8.3.3	ClearStringInDataStruct()	73
4.8.3.4	DisplayFullLine() [1/2]	74
4.8.3.5	DisplayFullLine() [2/2]	75
4.8.3.6	FindFirstCharAddr()	76
4.8.3.7	getDisplayDataPtr()	76
4.8.3.8	IsLineEmpty()	77
4.8.3.9	RefreshLine()	77
4.8.3.10	setLineAlignment()	78
4.8.3.11	setLineAlignmentAndRefresh()	79
4.8.3.12	shiftLine_task()	79
4.8.3.13	updateLineAndRefresh()	80
4.8.4	Member Data Documentation	81
4.8.4.1	display_data	81
4.8.4.2	dummy	81
4.8.4.3	isShiftInProgress	81
4.8.4.4	p_lcd	82
4.9	DisplayManagement Class Reference	82
4.9.1	Detailed Description	83
4.9.2	Constructor & Destructor Documentation	83
4.9.2.1	DisplayManagement()	83
4.9.3	Member Function Documentation	84
4.9.3.1	DisplaySensorData_Task()	84
4.9.3.2	GetIftPointer()	85
4.9.3.3	GetSensorMgtPtr()	85
4.9.3.4	RemoveWelcomeMessage_Task()	86
4.9.4	Member Data Documentation	86
4.9.4.1	p_display_ift	86
4.9.4.2	p_SensorMgt	87

4.10 HumSensor Class Reference . . . . .	87
4.10.1 Detailed Description . . . . .	88
4.10.2 Constructor & Destructor Documentation . . . . .	88
4.10.2.1 HumSensor() [1/2] . . . . .	88
4.10.2.2 HumSensor() [2/2] . . . . .	89
4.10.3 Member Function Documentation . . . . .	89
4.10.3.1 readHumSensor_task() . . . . .	89
4.10.3.2 updateTaskPeriod() . . . . .	90
4.11 I2C Class Reference . . . . .	91
4.11.1 Detailed Description . . . . .	92
4.11.2 Constructor & Destructor Documentation . . . . .	92
4.11.2.1 I2C() . . . . .	92
4.11.3 Member Function Documentation . . . . .	92
4.11.3.1 initializeBus() . . . . .	93
4.11.3.2 read() . . . . .	93
4.11.3.3 setBitRate() . . . . .	94
4.11.3.4 write() . . . . .	94
4.11.3.5 writeByte() . . . . .	95
4.11.4 Member Data Documentation . . . . .	96
4.11.4.1 bitrate . . . . .	96
4.12 keepAliveLed Class Reference . . . . .	96
4.12.1 Detailed Description . . . . .	97
4.12.2 Constructor & Destructor Documentation . . . . .	97
4.12.2.1 keepAliveLed() . . . . .	97
4.12.3 Member Function Documentation . . . . .	97
4.12.3.1 blinkLed_task() . . . . .	98
4.13 LCD Class Reference . . . . .	98
4.13.1 Detailed Description . . . . .	100
4.13.2 Constructor & Destructor Documentation . . . . .	100
4.13.2.1 LCD() . . . . .	100

---

4.13.3 Member Function Documentation . . . . .	101
4.13.3.1 command() . . . . .	101
4.13.3.2 ConfigureBacklight() . . . . .	102
4.13.3.3 ConfigureCursorBlink() . . . . .	103
4.13.3.4 ConfigureCursorOnOff() . . . . .	103
4.13.3.5 ConfigureDisplayOnOff() . . . . .	104
4.13.3.6 ConfigureEntryModeDir() . . . . .	104
4.13.3.7 ConfigureEntryModeShift() . . . . .	105
4.13.3.8 ConfigureFontType() . . . . .	106
4.13.3.9 ConfigureI2CAddr() . . . . .	107
4.13.3.10 ConfigureLineNumber() . . . . .	107
4.13.3.11 GetDDRAMAddress() . . . . .	108
4.13.3.12 GetLineNumberCnf() . . . . .	108
4.13.3.13 InitializeScreen() . . . . .	109
4.13.3.14 SetDDRAMAddress() . . . . .	109
4.13.3.15 write() . . . . .	110
4.13.3.16 write4bits() . . . . .	111
4.13.3.17 WriteInRam() . . . . .	112
4.13.4 Member Data Documentation . . . . .	113
4.13.4.1 backlight_enable . . . . .	113
4.13.4.2 cnfCursorBlink . . . . .	113
4.13.4.3 cnfCursorOnOff . . . . .	113
4.13.4.4 cnfDisplayOnOff . . . . .	113
4.13.4.5 cnfEntryModeDir . . . . .	113
4.13.4.6 cnfEntryModeShift . . . . .	114
4.13.4.7 cnfFontType . . . . .	114
4.13.4.8 cnfI2C_addr . . . . .	114
4.13.4.9 cnfLineNumber . . . . .	114
4.13.4.10 ddram_addr . . . . .	114
4.13.4.11 i2c_drv_ptr . . . . .	115

4.14 LinkedList Class Reference . . . . .	115
4.14.1 Detailed Description . . . . .	116
4.14.2 Member Typedef Documentation . . . . .	116
4.14.2.1 T_LL_element . . . . .	117
4.14.3 Constructor & Destructor Documentation . . . . .	117
4.14.3.1 LinkedList() . . . . .	117
4.14.3.2 ~LinkedList() . . . . .	117
4.14.4 Member Function Documentation . . . . .	118
4.14.4.1 AttachNewElement() . . . . .	118
4.14.4.2 FindElement() . . . . .	118
4.14.4.3 getCurrentElement() . . . . .	119
4.14.4.4 IsLLEmpty() . . . . .	120
4.14.4.5 MoveToNextElement() . . . . .	120
4.14.4.6 RemoveElement() . . . . .	121
4.14.4.7 ResetElementPtr() . . . . .	121
4.14.5 Member Data Documentation . . . . .	122
4.14.5.1 curElement_ptr . . . . .	122
4.14.5.2 firstElement . . . . .	122
4.15 PressSensor Class Reference . . . . .	122
4.15.1 Detailed Description . . . . .	123
4.15.2 Constructor & Destructor Documentation . . . . .	123
4.15.2.1 PressSensor() [1/2] . . . . .	124
4.15.2.2 PressSensor() [2/2] . . . . .	124
4.15.3 Member Function Documentation . . . . .	125
4.15.3.1 readPressSensor_task() . . . . .	125
4.15.3.2 updateTaskPeriod() . . . . .	126
4.16 scheduler Class Reference . . . . .	127
4.16.1 Detailed Description . . . . .	128
4.16.2 Member Typedef Documentation . . . . .	128
4.16.2.1 Task_t . . . . .	128

---

4.16.3 Constructor & Destructor Documentation . . . . .	128
4.16.3.1 scheduler() . . . . .	129
4.16.4 Member Function Documentation . . . . .	129
4.16.4.1 addPeriodicTask() . . . . .	129
4.16.4.2 getPitNumber() . . . . .	130
4.16.4.3 getTaskCount() . . . . .	131
4.16.4.4 launchPeriodicTasks() . . . . .	131
4.16.4.5 LLElementCompare() . . . . .	132
4.16.4.6 removePeriodicTask() . . . . .	133
4.16.4.7 startScheduling() . . . . .	133
4.16.4.8 updateTaskPeriod() . . . . .	134
4.16.5 Member Data Documentation . . . . .	135
4.16.5.1 pit_number . . . . .	135
4.16.5.2 task_count . . . . .	135
4.16.5.3 TasksLL_ptr . . . . .	136
4.17 Sensor Class Reference . . . . .	136
4.17.1 Detailed Description . . . . .	137
4.17.2 Constructor & Destructor Documentation . . . . .	137
4.17.2.1 Sensor() [1/2] . . . . .	137
4.17.2.2 Sensor() [2/2] . . . . .	138
4.17.3 Member Function Documentation . . . . .	138
4.17.3.1 getRawDataPtr() . . . . .	138
4.17.3.2 getTaskPeriod() . . . . .	139
4.17.3.3 getValidity() . . . . .	139
4.17.3.4 getValue() . . . . .	139
4.17.3.5 getValueDecimal() . . . . .	140
4.17.3.6 getValueInteger() . . . . .	140
4.17.3.7 readSensor_task() . . . . .	140
4.17.3.8 setLastValidity() . . . . .	140
4.17.3.9 setValidityTMO() . . . . .	141

---

4.17.3.10 updateTaskPeriod()	142
4.17.3.11 updateValidData()	142
4.17.4 Member Data Documentation	143
4.17.4.1 raw_data	143
4.17.4.2 task_period	143
4.17.4.3 valid_pit	143
4.17.4.4 valid_value	143
4.17.4.5 validity	143
4.17.4.6 validity_last_read	144
4.17.4.7 validity_tmo	144
4.18 SensorManagement Class Reference	144
4.18.1 Detailed Description	145
4.18.2 Constructor & Destructor Documentation	145
4.18.2.1 SensorManagement()	145
4.18.3 Member Function Documentation	145
4.18.3.1 getFullStringFormattedValue()	145
4.18.3.2 getSensorCount()	146
4.18.3.3 getSensorObjectPtr()	147
4.18.3.4 updateTaskPeriod()	148
4.18.4 Member Data Documentation	148
4.18.4.1 nb_sensors	149
4.18.4.2 sensor_ptr_table	149
4.19 String Class Reference	149
4.19.1 Detailed Description	150
4.19.2 Constructor & Destructor Documentation	150
4.19.2.1 String() [1/2]	150
4.19.2.2 String() [2/2]	151
4.19.2.3 ~String()	151
4.19.3 Member Function Documentation	152
4.19.3.1 appendBool()	152

---

4.19.3.2 appendChar() . . . . .	153
4.19.3.3 appendInteger() . . . . .	153
4.19.3.4 appendSpace() . . . . .	154
4.19.3.5 appendString() . . . . .	155
4.19.3.6 Clear() . . . . .	156
4.19.3.7 ComputeStringSize() . . . . .	157
4.19.3.8 getSize() . . . . .	157
4.19.3.9 getString() . . . . .	158
4.19.4 Member Data Documentation . . . . .	158
4.19.4.1 size . . . . .	158
4.19.4.2 string . . . . .	158
4.20 T_ASW_init_cnf Struct Reference . . . . .	159
4.20.1 Detailed Description . . . . .	159
4.20.2 Member Data Documentation . . . . .	159
4.20.2.1 isDebugActivated . . . . .	159
4.20.2.2 isDisplayActivated . . . . .	159
4.20.2.3 isLEDActivated . . . . .	160
4.20.2.4 isSensorMgtActivated . . . . .	160
4.21 Bmp180::T_BMP180_calib_data Struct Reference . . . . .	160
4.21.1 Detailed Description . . . . .	160
4.21.2 Member Data Documentation . . . . .	161
4.21.2.1 AC1 . . . . .	161
4.21.2.2 AC2 . . . . .	161
4.21.2.3 AC3 . . . . .	161
4.21.2.4 AC4 . . . . .	161
4.21.2.5 AC5 . . . . .	161
4.21.2.6 AC6 . . . . .	162
4.21.2.7 B1 . . . . .	162
4.21.2.8 B2 . . . . .	162
4.21.2.9 MB . . . . .	162

4.21.2.10 MC . . . . .	162
4.21.2.11 MD . . . . .	162
4.22 Bmp180::T_BMP180_measurement_data Struct Reference . . . . .	163
4.22.1 Detailed Description . . . . .	163
4.22.2 Member Data Documentation . . . . .	163
4.22.2.1 ready . . . . .	163
4.22.2.2 ts . . . . .	163
4.22.2.3 value . . . . .	163
4.23 T_display_data Struct Reference . . . . .	164
4.23.1 Detailed Description . . . . .	164
4.23.2 Member Data Documentation . . . . .	164
4.23.2.1 alignment . . . . .	165
4.23.2.2 display_str . . . . .	165
4.23.2.3 isEmpty . . . . .	165
4.23.2.4 mode . . . . .	165
4.23.2.5 shift_data . . . . .	165
4.24 T_Display_shift_data Struct Reference . . . . .	166
4.24.1 Detailed Description . . . . .	166
4.24.2 Member Data Documentation . . . . .	166
4.24.2.1 str_cur_ptr . . . . .	166
4.24.2.2 str_ptr . . . . .	167
4.24.2.3 temporization . . . . .	167
4.25 T_LCD_conf_struct Struct Reference . . . . .	167
4.25.1 Detailed Description . . . . .	167
4.25.2 Member Data Documentation . . . . .	168
4.25.2.1 backlight_en . . . . .	168
4.25.2.2 cursor_en . . . . .	168
4.25.2.3 cursorBlink_en . . . . .	168
4.25.2.4 display_en . . . . .	168
4.25.2.5 entryModeDir . . . . .	168

4.25.2.6	entryModeShift . . . . .	169
4.25.2.7	fontType_cnf . . . . .	169
4.25.2.8	i2c_addr . . . . .	169
4.25.2.9	i2c_bitrate . . . . .	169
4.25.2.10	lineNumber_cnf . . . . .	169
4.26	LinkedList::T_LL_element Struct Reference . . . . .	170
4.26.1	Detailed Description . . . . .	170
4.26.2	Member Data Documentation . . . . .	170
4.26.2.1	data_ptr . . . . .	170
4.26.2.2	nextElement . . . . .	170
4.27	T_SensorManagement_Sensor_Config Struct Reference . . . . .	171
4.27.1	Detailed Description . . . . .	171
4.27.2	Member Data Documentation . . . . .	171
4.27.2.1	data_name_str . . . . .	171
4.27.2.2	period . . . . .	171
4.27.2.3	sensor_type . . . . .	171
4.27.2.4	unit_str . . . . .	172
4.27.2.5	validity_tmo . . . . .	172
4.28	scheduler::Task_t Struct Reference . . . . .	172
4.28.1	Detailed Description . . . . .	172
4.28.2	Member Data Documentation . . . . .	172
4.28.2.1	period . . . . .	172
4.28.2.2	TaskPtr . . . . .	173
4.29	TempSensor Class Reference . . . . .	173
4.29.1	Detailed Description . . . . .	174
4.29.2	Constructor & Destructor Documentation . . . . .	174
4.29.2.1	TempSensor() [1/2] . . . . .	174
4.29.2.2	TempSensor() [2/2] . . . . .	175
4.29.3	Member Function Documentation . . . . .	176
4.29.3.1	readTempSensor_task() . . . . .	176

4.29.3.2 updateTaskPeriod()	176
4.30 timer Class Reference	177
4.30.1 Detailed Description	178
4.30.2 Constructor & Destructor Documentation	178
4.30.2.1 timer()	178
4.30.3 Member Function Documentation	178
4.30.3.1 configureTimer1()	178
4.30.3.2 configureTimer3()	179
4.30.3.3 getTimer1Value()	180
4.30.3.4 startTimer1()	180
4.30.3.5 startTimer3()	181
4.30.3.6 stopTimer1()	181
4.30.3.7 stopTimer3()	182
4.30.4 Member Data Documentation	182
4.30.4.1 prescaler1	182
4.30.4.2 prescaler3	182
4.31 usart Class Reference	183
4.31.1 Detailed Description	183
4.31.2 Constructor & Destructor Documentation	183
4.31.2.1 usart()	183
4.31.3 Member Function Documentation	184
4.31.3.1 setBaudRate()	184
4.31.3.2 usart_init()	185
4.31.3.3 usart_read()	185
4.31.3.4 usart_sendByte()	185
4.31.3.5 usart_sendString()	186
4.31.3.6 usart_transmit()	187
4.31.4 Member Data Documentation	188
4.31.4.1 BaudRate	188
4.32 Watchdog Class Reference	188

4.32.1	Detailed Description	189
4.32.2	Constructor & Destructor Documentation	189
4.32.2.1	Watchdog() [1/2]	189
4.32.2.2	Watchdog() [2/2]	190
4.32.3	Member Function Documentation	190
4.32.3.1	disable()	190
4.32.3.2	enable()	191
4.32.3.3	getTMOValue()	192
4.32.3.4	isEnabled()	192
4.32.3.5	reset()	193
4.32.3.6	SwitchWdg()	193
4.32.3.7	SystemReset()	194
4.32.3.8	timeoutUpdate()	195
4.32.4	Member Data Documentation	195
4.32.4.1	isActive	195
4.32.4.2	tmo_value	196
<b>5</b>	<b>File Documentation</b>	<b>197</b>
5.1	asw.cpp File Reference	197
5.1.1	Detailed Description	198
5.1.2	Function Documentation	198
5.1.2.1	asw_init()	198
5.2	asw.h File Reference	199
5.2.1	Detailed Description	199
5.2.2	Function Documentation	199
5.2.2.1	asw_init()	200
5.3	Bmp180.cpp File Reference	200
5.3.1	Detailed Description	201
5.3.2	Variable Documentation	201
5.3.2.1	p_global_BSW_bmp180	201
5.4	Bmp180.h File Reference	201

5.4.1	Detailed Description . . . . .	202
5.4.2	Macro Definition Documentation . . . . .	203
5.4.2.1	BMP180_CHIP_ID_CALIB_EEP_START_ADDR . . . . .	203
5.4.2.2	BMP180_CHIP_ID_EEP_ADDR . . . . .	203
5.4.2.3	BMP180_CHIP_ID_EXPECTED . . . . .	203
5.4.2.4	BMP180_CTRL_MEAS_EEP_ADDR . . . . .	203
5.4.2.5	BMP180_CTRL_MEAS_START_PRESS_CONV_OSS0 . . . . .	203
5.4.2.6	BMP180_CTRL_MEAS_START_TEMP_CONV . . . . .	204
5.4.2.7	BMP180_I2C_ADDR . . . . .	204
5.4.2.8	BMP180_I2C_BITRATE . . . . .	204
5.4.2.9	BMP180_MONITORING_DEFAULT_PERIOD . . . . .	204
5.4.2.10	BMP180_OUT_REG_LSB EEPROM_ADDR . . . . .	204
5.4.2.11	BMP180_OUT_REG_MSB EEPROM_ADDR . . . . .	205
5.4.2.12	BMP180_PRESS_MEAS_OSS0_TIMER_CTC_VALUE . . . . .	205
5.4.2.13	BMP180_PRESS_MEAS_OSS0_WAITING_TIME . . . . .	205
5.4.2.14	BMP180_TEMP_MEAS_TIMER_CTC_VALUE . . . . .	205
5.4.2.15	BMP180_TEMP_MEAS_WAITING_TIME . . . . .	205
5.4.2.16	BMP180_TIMER_PRESCALER_VALUE . . . . .	206
5.4.3	Enumeration Type Documentation . . . . .	206
5.4.3.1	T_BMP180_status . . . . .	206
5.4.4	Variable Documentation . . . . .	206
5.4.4.1	p_global_BSW_bmp180 . . . . .	206
5.5	bsw.cpp File Reference . . . . .	206
5.5.1	Detailed Description . . . . .	207
5.5.2	Function Documentation . . . . .	207
5.5.2.1	bsw_init() . . . . .	207
5.6	bsw.h File Reference . . . . .	208
5.6.1	Detailed Description . . . . .	208
5.6.2	Function Documentation . . . . .	209
5.6.2.1	bsw_init() . . . . .	209

5.7	CpuLoad.cpp File Reference . . . . .	209
5.7.1	Detailed Description . . . . .	210
5.7.2	Variable Documentation . . . . .	210
5.7.2.1	p_global_BSW_cpupload . . . . .	210
5.8	CpuLoad.h File Reference . . . . .	210
5.8.1	Detailed Description . . . . .	211
5.8.2	Macro Definition Documentation . . . . .	211
5.8.2.1	NB_OF_SAMPLES . . . . .	211
5.8.3	Variable Documentation . . . . .	211
5.8.3.1	p_global_BSW_cpupload . . . . .	211
5.9	DebugInterface.cpp File Reference . . . . .	212
5.9.1	Detailed Description . . . . .	212
5.9.2	Variable Documentation . . . . .	212
5.9.2.1	p_global_ASW_DebugInterface . . . . .	212
5.10	DebugInterface.h File Reference . . . . .	213
5.10.1	Detailed Description . . . . .	213
5.10.2	Macro Definition Documentation . . . . .	213
5.10.2.1	USART_BAUDRATE . . . . .	214
5.10.3	Variable Documentation . . . . .	214
5.10.3.1	p_global_ASW_DebugInterface . . . . .	214
5.11	DebugManagement.cpp File Reference . . . . .	214
5.11.1	Detailed Description . . . . .	215
5.11.2	Variable Documentation . . . . .	215
5.11.2.1	p_global_ASW_DebugManagement . . . . .	215
5.11.2.2	str_debug_info_message_wdg_disabled . . . . .	216
5.11.2.3	str_debug_info_message_wdg_enabled . . . . .	216
5.11.2.4	str_debug_info_message_wdg_tmo_updated . . . . .	216
5.11.2.5	str_debug_info_message_wdg_tmo_value . . . . .	216
5.11.2.6	str_debug_info_message_wrong_menu_selection . . . . .	216
5.11.2.7	str_debug_main_menu . . . . .	217

---

5.11.2.8 str_debug_wdg_menu . . . . .	217
5.11.2.9 str_debug_wdg_timeout_update_selection . . . . .	217
5.12 DebugManagement.h File Reference . . . . .	218
5.12.1 Detailed Description . . . . .	218
5.12.2 Macro Definition Documentation . . . . .	219
5.12.2.1 PERIOD_MS_TASK_DISPLAY_CPU_LOAD . . . . .	219
5.12.2.2 PERIOD_MS_TASK_DISPLAY_DEBUG_DATA . . . . .	219
5.12.3 Enumeration Type Documentation . . . . .	219
5.12.3.1 debug_mgt_main_menu_state_t . . . . .	219
5.12.3.2 debug_mgt_wdg_state_t . . . . .	219
5.12.4 Variable Documentation . . . . .	220
5.12.4.1 p_global_ASW_DebugManagement . . . . .	220
5.13 dht22.cpp File Reference . . . . .	220
5.13.1 Detailed Description . . . . .	221
5.13.2 Macro Definition Documentation . . . . .	221
5.13.2.1 MAX_WAIT_TIME_US . . . . .	221
5.13.3 Variable Documentation . . . . .	221
5.13.3.1 p_global_BSW_dht22 . . . . .	221
5.14 dht22.h File Reference . . . . .	222
5.14.1 Detailed Description . . . . .	222
5.14.2 Variable Documentation . . . . .	222
5.14.2.1 p_global_BSW_dht22 . . . . .	222
5.15 dio.cpp File Reference . . . . .	223
5.15.1 Detailed Description . . . . .	223
5.15.2 Variable Documentation . . . . .	223
5.15.2.1 p_global_BSW_dio . . . . .	224
5.16 dio.h File Reference . . . . .	224
5.16.1 Detailed Description . . . . .	225
5.16.2 Macro Definition Documentation . . . . .	225
5.16.2.1 DECODE_PIN . . . . .	225

---

5.16.2.2 DECODE_PORT . . . . .	225
5.16.2.3 ENCODE_PORT . . . . .	225
5.16.2.4 PORT_CNF_IN . . . . .	226
5.16.2.5 PORT_CNF_OUT . . . . .	226
5.16.3 Variable Documentation . . . . .	226
5.16.3.1 p_global_BSW_dio . . . . .	226
5.17 dio_port_cnf.h File Reference . . . . .	226
5.17.1 Detailed Description . . . . .	227
5.17.2 Macro Definition Documentation . . . . .	227
5.17.2.1 PORT_A . . . . .	227
5.17.2.2 PORT_B . . . . .	227
5.17.2.3 PORT_C . . . . .	228
5.17.2.4 PORT_D . . . . .	228
5.17.2.5 PORTB_CNF_DDRB . . . . .	228
5.17.2.6 PORTB_CNF_PORTB . . . . .	228
5.18 dio_reg_atm2560.h File Reference . . . . .	229
5.18.1 Detailed Description . . . . .	229
5.18.2 Macro Definition Documentation . . . . .	229
5.18.2.1 DDRA_PTR . . . . .	230
5.18.2.2 DDRB_PTR . . . . .	230
5.18.2.3 DDRC_PTR . . . . .	230
5.18.2.4 DDRD_PTR . . . . .	230
5.18.2.5 PINA_PTR . . . . .	230
5.18.2.6 PINB_PTR . . . . .	231
5.18.2.7 PINC_PTR . . . . .	231
5.18.2.8 PIND_PTR . . . . .	231
5.18.2.9 PORTA_PTR . . . . .	231
5.18.2.10 PORTB_PTR . . . . .	231
5.18.2.11 PORTC_PTR . . . . .	232
5.18.2.12 PORTD_PTR . . . . .	232

5.19 DisplayInterface.cpp File Reference . . . . .	232
5.19.1 Detailed Description . . . . .	232
5.19.2 Variable Documentation . . . . .	233
5.19.2.1 p_global_ASW_DisplayInterface . . . . .	233
5.20 DisplayInterface.h File Reference . . . . .	233
5.20.1 Detailed Description . . . . .	234
5.20.2 Macro Definition Documentation . . . . .	234
5.20.2.1 DISPLAY_LINE_SHIFT_PERIOD_MS . . . . .	234
5.20.2.2 DISPLAY_LINE_SHIFT_TEMPO_TIME . . . . .	234
5.20.3 Enumeration Type Documentation . . . . .	234
5.20.3.1 T_DisplayInterface_LineAlignment . . . . .	234
5.20.3.2 T_DisplayInterface_LineDisplayMode . . . . .	235
5.20.4 Variable Documentation . . . . .	235
5.20.4.1 p_global_ASW_DisplayInterface . . . . .	235
5.21 DisplayManagement.cpp File Reference . . . . .	236
5.21.1 Detailed Description . . . . .	236
5.21.2 Variable Documentation . . . . .	236
5.21.2.1 noSensorsDisplayString . . . . .	236
5.21.2.2 p_global_ASW_DisplayManagement . . . . .	237
5.21.2.3 welcomeMessageString . . . . .	237
5.22 DisplayManagement.h File Reference . . . . .	237
5.22.1 Detailed Description . . . . .	238
5.22.2 Macro Definition Documentation . . . . .	238
5.22.2.1 DISPLAY_MGT_FIRST_LINE_SENSORS . . . . .	238
5.22.2.2 DISPLAY_MGT_I2C_BITRATE . . . . .	238
5.22.2.3 DISPLAY_MGT_LCD_I2C_ADDR . . . . .	238
5.22.2.4 DISPLAY_MGT_PERIOD_TASK_SENSOR . . . . .	239
5.22.2.5 DISPLAY_MGT_PERIOD_WELCOME_MSG_REMOVAL . . . . .	239
5.22.3 Variable Documentation . . . . .	239
5.22.3.1 LCD_init_cnf . . . . .	239

5.22.3.2 p_global_ASW_DisplayManagement . . . . .	239
5.23 HumSensor.cpp File Reference . . . . .	240
5.23.1 Detailed Description . . . . .	240
5.23.2 Macro Definition Documentation . . . . .	240
5.23.2.1 DHT22_PORT . . . . .	240
5.24 HumSensor.h File Reference . . . . .	241
5.24.1 Detailed Description . . . . .	241
5.25 I2C.cpp File Reference . . . . .	241
5.25.1 Detailed Description . . . . .	242
5.25.2 Variable Documentation . . . . .	242
5.25.2.1 p_global_BSW_i2c . . . . .	242
5.26 I2C.h File Reference . . . . .	242
5.26.1 Detailed Description . . . . .	243
5.26.2 Macro Definition Documentation . . . . .	243
5.26.2.1 DATA_ACK . . . . .	243
5.26.2.2 REPEATED_START . . . . .	243
5.26.2.3 SLAR_ACK . . . . .	244
5.26.2.4 SLAW_ACK . . . . .	244
5.26.2.5 START . . . . .	244
5.26.3 Variable Documentation . . . . .	244
5.26.3.1 p_global_BSW_i2c . . . . .	244
5.27 int.cpp File Reference . . . . .	245
5.27.1 Detailed Description . . . . .	245
5.27.2 Function Documentation . . . . .	245
5.27.2.1 ISR() [1/3] . . . . .	246
5.27.2.2 ISR() [2/3] . . . . .	246
5.27.2.3 ISR() [3/3] . . . . .	247
5.28 keepAliveLed.cpp File Reference . . . . .	248
5.28.1 Detailed Description . . . . .	248
5.28.2 Variable Documentation . . . . .	248

---

5.28.2.1 p_global_ASW_keepAliveLed . . . . .	248
5.29 keepAliveLed.h File Reference . . . . .	249
5.29.1 Detailed Description . . . . .	249
5.29.2 Macro Definition Documentation . . . . .	249
5.29.2.1 LED_PORT . . . . .	250
5.29.2.2 PERIOD_MS_TASK_LED . . . . .	250
5.29.3 Variable Documentation . . . . .	250
5.29.3.1 p_global_ASW_keepAliveLed . . . . .	250
5.30 LCD.cpp File Reference . . . . .	250
5.30.1 Detailed Description . . . . .	251
5.30.2 Variable Documentation . . . . .	251
5.30.2.1 p_global_BSW_lcd . . . . .	251
5.31 LCD.h File Reference . . . . .	251
5.31.1 Detailed Description . . . . .	253
5.31.2 Macro Definition Documentation . . . . .	253
5.31.2.1 BACKLIGHT_PIN . . . . .	253
5.31.2.2 EN_PIN . . . . .	253
5.31.2.3 LCD_CNF_BACKLIGHT_OFF . . . . .	253
5.31.2.4 LCD_CNF_BACKLIGHT_ON . . . . .	254
5.31.2.5 LCD_CNF_CURSOR_BLINK_OFF . . . . .	254
5.31.2.6 LCD_CNF_CURSOR_BLINK_ON . . . . .	254
5.31.2.7 LCD_CNF_CURSOR_OFF . . . . .	254
5.31.2.8 LCD_CNF_CURSOR_ON . . . . .	254
5.31.2.9 LCD_CNF_DISPLAY_OFF . . . . .	255
5.31.2.10 LCD_CNF_DISPLAY_ON . . . . .	255
5.31.2.11 LCD_CNF_ENTRY_MODE_DIRECTION_LEFT . . . . .	255
5.31.2.12 LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT . . . . .	255
5.31.2.13 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF . . . . .	255
5.31.2.14 LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON . . . . .	256
5.31.2.15 LCD_CNF_FONT_5_11 . . . . .	256

5.31.2.16 LCD_CNF_FONT_5_8 . . . . .	256
5.31.2.17 LCD_CNF_ONE_LINE . . . . .	256
5.31.2.18 LCD_CNF_SHIFT_ID . . . . .	256
5.31.2.19 LCD_CNF_SHIFT_SH . . . . .	257
5.31.2.20 LCD_CNF_TWO_LINE . . . . .	257
5.31.2.21 LCD_DISPLAY_CTRL_FIELD_B . . . . .	257
5.31.2.22 LCD_DISPLAY_CTRL_FIELD_C . . . . .	257
5.31.2.23 LCD_DISPLAY_CTRL_FIELD_D . . . . .	257
5.31.2.24 LCD_FCT_SET_FIELD_DL . . . . .	258
5.31.2.25 LCD_FCT_SET_FIELD_F . . . . .	258
5.31.2.26 LCD_FCT_SET_FIELD_N . . . . .	258
5.31.2.27 LCD_INST_CLR_DISPLAY_BIT . . . . .	258
5.31.2.28 LCD_INST_DISPLAY_CTRL . . . . .	258
5.31.2.29 LCD_INST_ENTRY_MODE_SET . . . . .	259
5.31.2.30 LCD_INST_FUNCTION_SET . . . . .	259
5.31.2.31 LCD_INST_SET_DDRAM_ADDR . . . . .	259
5.31.2.32 LCD_RAM_1_LINE_MAX . . . . .	259
5.31.2.33 LCD_RAM_1_LINE_MIN . . . . .	259
5.31.2.34 LCD_RAM_2_LINES_MAX_1 . . . . .	260
5.31.2.35 LCD_RAM_2_LINES_MAX_2 . . . . .	260
5.31.2.36 LCD_RAM_2_LINES_MIN_1 . . . . .	260
5.31.2.37 LCD_RAM_2_LINES_MIN_2 . . . . .	260
5.31.2.38 LCD_SIZE_NB_CHAR_PER_LINE . . . . .	260
5.31.2.39 LCD_SIZE_NB_LINES . . . . .	261
5.31.2.40 LCD_WAIT_CLR_RETURN . . . . .	261
5.31.2.41 LCD_WAIT_OTHER_MODES . . . . .	261
5.31.2.42 RS_PIN . . . . .	261
5.31.2.43 RW_PIN . . . . .	261
5.31.3 Enumeration Type Documentation . . . . .	261
5.31.3.1 T_LCD_command . . . . .	261

---

5.31.3.2 <code>T_LCD_config_mode</code>	262
5.31.3.3 <code>T_LCD_ram_area</code>	262
5.31.4 Variable Documentation	262
5.31.4.1 <code>p_global_BSW_lcd</code>	263
5.32 <code>LinkedList.cpp</code> File Reference	263
5.32.1 Detailed Description	263
5.33 <code>LinkedList.h</code> File Reference	263
5.33.1 Detailed Description	264
5.33.2 Typedef Documentation	264
5.33.2.1 <code>CompareFctPtr_t</code>	264
5.34 <code>main.cpp</code> File Reference	264
5.34.1 Detailed Description	265
5.34.2 Macro Definition Documentation	265
5.34.2.1 <code>DEBUG_ACTIVE_PORT</code>	265
5.34.3 Function Documentation	265
5.34.3.1 <code>main()</code>	266
5.34.4 Variable Documentation	266
5.34.4.1 <code>ASW_init_cnf</code>	266
5.34.4.2 <code>isDebugModeActivated</code>	266
5.35 <code>main.h</code> File Reference	267
5.35.1 Detailed Description	267
5.35.2 Variable Documentation	267
5.35.2.1 <code>ASW_init_cnf</code>	267
5.35.2.2 <code>isDebugModeActivated</code>	268
5.36 <code>operators.cpp</code> File Reference	268
5.36.1 Detailed Description	268
5.36.2 Function Documentation	269
5.36.2.1 <code>operator delete()</code>	269
5.36.2.2 <code>operator new()</code>	269
5.37 <code>operators.h</code> File Reference	269

5.37.1	Detailed Description	270
5.37.2	Function Documentation	270
5.37.2.1	operator delete()	270
5.37.2.2	operator new()	271
5.38	PressSensor.cpp File Reference	271
5.38.1	Detailed Description	272
5.39	PressSensor.h File Reference	272
5.39.1	Detailed Description	272
5.40	scheduler.cpp File Reference	273
5.40.1	Detailed Description	273
5.40.2	Variable Documentation	273
5.40.2.1	p_global_scheduler	273
5.41	scheduler.h File Reference	274
5.41.1	Detailed Description	274
5.41.2	Macro Definition Documentation	274
5.41.2.1	PRESCALER_PERIODIC_TIMER	275
5.41.2.2	SW_PERIOD_MS	275
5.41.2.3	TIMER_CTC_VALUE	275
5.41.3	Typedef Documentation	275
5.41.3.1	TaskPtr_t	275
5.41.4	Variable Documentation	275
5.41.4.1	p_global_scheduler	276
5.42	Sensor.cpp File Reference	276
5.42.1	Detailed Description	276
5.42.2	Macro Definition Documentation	276
5.42.2.1	TASK_PERIOD_DEFAULT	277
5.42.2.2	VALIDITY_TIMEOUT_MS_DEFAULT	277
5.43	Sensor.h File Reference	277
5.43.1	Detailed Description	277
5.44	sensor_configuration.cpp File Reference	278

5.44.1	Detailed Description	278
5.44.2	Macro Definition Documentation	278
5.44.2.1	SENSOR_MGT_CNF_DEFAULT_PERIOD	279
5.44.2.2	SENSOR_MGT_CNF_DEFAULT_TMO	279
5.44.3	Variable Documentation	279
5.44.3.1	SensorManagement_Sensor_Config_list	279
5.45	sensor_configuration.h File Reference	280
5.45.1	Detailed Description	280
5.45.2	Variable Documentation	280
5.45.2.1	SensorManagement_Sensor_Config_list	281
5.46	SensorManagement.cpp File Reference	281
5.46.1	Detailed Description	281
5.46.2	Variable Documentation	281
5.46.2.1	p_global_ASW_SensorManagement	282
5.47	SensorManagement.h File Reference	282
5.47.1	Detailed Description	282
5.47.2	Enumeration Type Documentation	282
5.47.2.1	T_SensorManagement_Sensor_Type	282
5.47.3	Variable Documentation	283
5.47.3.1	p_global_ASW_SensorManagement	283
5.48	String.cpp File Reference	283
5.48.1	Detailed Description	283
5.49	String.h File Reference	284
5.49.1	Detailed Description	284
5.50	TempSensor.cpp File Reference	284
5.50.1	Detailed Description	285
5.50.2	Macro Definition Documentation	285
5.50.2.1	DHT22_PORT	285
5.50.2.2	TEMP_SENSOR_VALUE_ACCEPTABLE_DIFFERENCE	285
5.51	TempSensor.h File Reference	285

---

5.51.1 Detailed Description . . . . .	286
5.52 timer.cpp File Reference . . . . .	286
5.52.1 Detailed Description . . . . .	286
5.52.2 Variable Documentation . . . . .	287
5.52.2.1 p_global_BSW_timer . . . . .	287
5.53 timer.h File Reference . . . . .	287
5.53.1 Detailed Description . . . . .	287
5.53.2 Variable Documentation . . . . .	288
5.53.2.1 p_global_BSW_timer . . . . .	288
5.54 usart.cpp File Reference . . . . .	288
5.54.1 Detailed Description . . . . .	288
5.54.2 Variable Documentation . . . . .	289
5.54.2.1 p_global_BSW_usart . . . . .	289
5.55 usart.h File Reference . . . . .	289
5.55.1 Detailed Description . . . . .	289
5.55.2 Variable Documentation . . . . .	290
5.55.2.1 p_global_BSW_usart . . . . .	290
5.56 Watchdog.cpp File Reference . . . . .	290
5.56.1 Detailed Description . . . . .	291
5.56.2 Macro Definition Documentation . . . . .	291
5.56.2.1 WDG_TIMEOUT_DEFAULT_MS . . . . .	291
5.56.3 Variable Documentation . . . . .	291
5.56.3.1 p_global_BSW_wdg . . . . .	291
5.57 Watchdog.h File Reference . . . . .	291
5.57.1 Detailed Description . . . . .	292
5.57.2 Macro Definition Documentation . . . . .	292
5.57.2.1 WDG_TMO_120MS . . . . .	292
5.57.2.2 WDG_TMO_15MS . . . . .	293
5.57.2.3 WDG_TMO_1S . . . . .	293
5.57.2.4 WDG_TMO_250MS . . . . .	293
5.57.2.5 WDG_TMO_2S . . . . .	293
5.57.2.6 WDG_TMO_30MS . . . . .	293
5.57.2.7 WDG_TMO_4S . . . . .	294
5.57.2.8 WDG_TMO_500MS . . . . .	294
5.57.2.9 WDG_TMO_60MS . . . . .	294
5.57.2.10 WDG_TMO_8S . . . . .	294
5.57.3 Variable Documentation . . . . .	294
5.57.3.1 p_global_BSW_wdg . . . . .	294
Index . . . . .	295

# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

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

Bmp180 . . . . .	9
CpuLoad . . . . .	28
debug_mgt_state_struct_t . . . . .	33
DebugInterface . . . . .	34
DebugManagement . . . . .	41
dht22 . . . . .	53
dio . . . . .	60
DisplayInterface . . . . .	69
DisplayManagement . . . . .	82
I2C . . . . .	91
keepAliveLed . . . . .	96
LCD . . . . .	98
LinkedList . . . . .	115
scheduler . . . . .	127
Sensor . . . . .	136
HumSensor . . . . .	87
PressSensor . . . . .	122
TempSensor . . . . .	173
SensorManagement . . . . .	144
String . . . . .	149
T_ASW_init_cnf . . . . .	159
Bmp180::T_BMP180_calib_data . . . . .	160
Bmp180::T_BMP180_measurement_data . . . . .	163
T_display_data . . . . .	164
T_Display_shift_data . . . . .	166
T_LCD_conf_struct . . . . .	167
LinkedList::T_LL_element . . . . .	170
T_SensorManagement_Sensor_Config . . . . .	171
scheduler::Task_t . . . . .	172
timer . . . . .	177
uart . . . . .	183
Watchdog . . . . .	188



# Chapter 2

## Class Index

### 2.1 Class List

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

<a href="#">Bmp180</a>	BMP180 sensor class definition	9
<a href="#">CpuLoad</a>	Class defining CPU load libraries	28
<a href="#">debug_mgt_state_struct_t</a>	Structure containing all debug states	33
<a href="#">DebugInterface</a>	Class used for debugging on usart link	34
<a href="#">DebugManagement</a>	Debug management class	41
<a href="#">dht22</a>	DHT 22 driver class	53
<a href="#">dio</a>	DIO class	60
<a href="#">DisplayInterface</a>	Display interface services class	69
<a href="#">DisplayManagement</a>	Display management class	82
<a href="#">HumSensor</a>	Class for humidity sensor	87
<a href="#">I2C</a>	Two-wire serial interface ( <a href="#">I2C</a> ) class definition	91
<a href="#">keepAliveLed</a>	Class for keep-alive LED blinking	96
<a href="#">LCD</a>	Class for <a href="#">LCD</a> S2004A display driver	98
<a href="#">LinkedList</a>	Linked list class	115
<a href="#">PressSensor</a>	Class for pressure sensor	122
<a href="#">scheduler</a>	Scheduler class	127
<a href="#">Sensor</a>	Generic class for sensor device	136
<a href="#">SensorManagement</a>	<a href="#">Sensor</a> management class	144

String		
String	management class . . . . .	149
T_ASW_init_cnf		
ASW initialization configuration structure . . . . .	159	
Bmp180::T_BMP180_calib_data		
Structure defining the calibration data of BMP180 sensor . . . . .	160	
Bmp180::T_BMP180_measurement_data		
Structure defining a sensor value and its status . . . . .	163	
T_display_data		
Structure containing display data . . . . .	164	
T_Display_shift_data		
Structure containing shift data . . . . .	166	
T_LCD_conf_struct		
Structure defining LCD configuration . . . . .	167	
LinkedList::T_LL_element		
Type defining a linked list element . . . . .	170	
T_SensorManagement_Sensor_Config		
Sensor informations structure . . . . .	171	
scheduler::Task_t		
Type defining a task structure . . . . .	172	
TempSensor		
Class for temperature sensor . . . . .	173	
timer		
Class defining a timer . . . . .	177	
uart		
USART serial bus class . . . . .	183	
Watchdog		
Watchdog management class . . . . .	188	

# Chapter 3

## File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

<a href="#">asw.cpp</a>	ASW main file . . . . .	197
<a href="#">asw.h</a>	ASW main header file . . . . .	199
<a href="#">Bmp180.cpp</a>	Bmp180 class source file . . . . .	200
<a href="#">Bmp180.h</a>	Bmp180 class header file . . . . .	201
<a href="#">bsw.cpp</a>	BSW main file . . . . .	206
<a href="#">bsw.h</a>	BSW main header file . . . . .	208
<a href="#">CpuLoad.cpp</a>	Defines functions of class CpuLoad . . . . .	209
<a href="#">CpuLoad.h</a>	CpuLoad class header file . . . . .	210
<a href="#">DebugInterface.cpp</a>	This file defines classes for log and debug data transmission on USART link . . . . .	212
<a href="#">DebugInterface.h</a>	Header file for debug and logging functions . . . . .	213
<a href="#">DebugManagement.cpp</a>	Debug management class source file . . . . .	214
<a href="#">DebugManagement.h</a>	Debug management class header file . . . . .	218
<a href="#">dht22.cpp</a>	This file defines classes for DHT22 driver . . . . .	220
<a href="#">dht22.h</a>	DHT22 driver header file . . . . .	222
<a href="#">dio.cpp</a>	DIO library . . . . .	223
<a href="#">dio.h</a>	DIO library header file . . . . .	224
<a href="#">dio_port_cnf.h</a>	Digital ports configuration file . . . . .	226
<a href="#">dio_reg_atm2560.h</a>	Defines DIO register addresses for ATMEGA2560 . . . . .	229

DisplayInterface.cpp	Source code file for display services . . . . .	232
DisplayInterface.h	DisplayInterface class header file . . . . .	233
DisplayManagement.cpp	Display management source file . . . . .	236
DisplayManagement.h	Display management class header file . . . . .	237
HumSensor.cpp	Defines function of class HumSensor . . . . .	240
HumSensor.h	Class HumSensor header file . . . . .	241
I2C.cpp	Two-wire interface (I2C) source file . . . . .	241
I2C.h	I2C class header file . . . . .	242
int.cpp	Interrupt management source file . . . . .	245
keepAliveLed.cpp	Definition of function for class keepAliveLed . . . . .	248
keepAliveLed.h	Class keepAliveLed header file . . . . .	249
LCD.cpp	LCD class source file . . . . .	250
LCD.h	LCD class header file . . . . .	251
LinkedList.cpp	Linked List library source file . . . . .	263
LinkedList.h	Linked List library header file . . . . .	263
main.cpp	Background task file . . . . .	264
main.h	Background task header file . . . . .	267
operators.cpp	C++ operators definitions . . . . .	268
operators.h	C++ operators definitions header file . . . . .	269
PressSensor.cpp	Defines function of class PressSensor . . . . .	271
PressSensor.h	Class PressSensor header file . . . . .	272
scheduler.cpp	Defines scheduler class . . . . .	273
scheduler.h	Scheduler class header file . . . . .	274
Sensor.cpp	Sensor class source code file . . . . .	276
Sensor.h	Sensor class header file . . . . .	277
sensor_configuration.cpp	Sensor configuration file . . . . .	278
sensor_configuration.h	Sensors configuration header file . . . . .	280
SensorManagement.cpp	SensorManagement class source code file . . . . .	281
SensorManagement.h	SensorManagement class header file . . . . .	282

<a href="#">String.cpp</a>		
<a href="#">String</a> class source file	.....	283
<a href="#">String.h</a>		
<a href="#">String</a> class header file	.....	284
<a href="#">TempSensor.cpp</a>		
Defines function of class <a href="#">TempSensor</a>	.....	284
<a href="#">TempSensor.h</a>		
Class <a href="#">TempSensor</a> header file	.....	285
<a href="#">timer.cpp</a>		
Defines function for class timer	.....	286
<a href="#">timer.h</a>		
Timer class header file	.....	287
<a href="#">usart.cpp</a>		
BSW library for USART	.....	288
<a href="#">usart.h</a>		
Header file for USART library	.....	289
<a href="#">Watchdog.cpp</a>		
Class <a href="#">Watchdog</a> source code file	.....	290
<a href="#">Watchdog.h</a>		
Class <a href="#">Watchdog</a> header file	.....	291



# Chapter 4

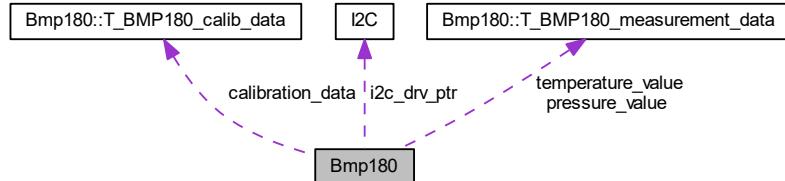
## Class Documentation

### 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.*
- bool [getPressureValue \(uint16\\_t \\*data\)](#)  
*Pressure reading function.*

- void `conversionTimerInterrupt ()`  
*End of conversion interrupt.*
- `T_BMP180_status getStatus ()`  
*Driver status get function.*
- `uint16_t getMonitoringTaskPeriod ()`  
*Task period get function.*
- void `startNewTemperatureConversion ()`  
*Starts a new temperature conversion.*
- void `startNewPressureConversion ()`  
*Starts a new pressure conversion.*
- void `TemperatureMonitoring ()`  
*Temperature value monitoring function.*
- void `PressureMonitoring ()`  
*Pressure value monitoring function.*
- void `ActivateTemperatureConversion (uint16_t req_period)`  
*Temperature conversion activation function.*
- void `ActivatePressureConversion (uint16_t req_period)`  
*Pressure conversion activation function.*
- void `StopTemperatureConversion ()`  
*Temperature conversion stop function.*
- void `StopPressureConversion ()`  
*Pressure conversion stop function.*
- bool `isTempConversionActivated ()`  
*Temperature conversion activation flag get function.*
- bool `isPressConversionActivated ()`  
*Pressure conversion activation flag get function.*

## Static Public Member Functions

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

## Private Member Functions

- void `readCalibData ()`  
*Calibration data reading function.*
- void `readChipID ()`  
*Chip ID read function.*
- void `CalculateTemperature (uint16_t UT)`  
*Temperature calculation function.*
- void `CalculatePressure (uint32_t UP)`  
*Pressure calculation function.*

## Private Attributes

- `I2C * i2c_drv_ptr`
- `uint8_t chip_id`
- `T_BMP180_status status`
- `uint16_t task_period`
- `bool isTempConvActivated`
- `bool isPressConvActivated`
- `T_BMP180_calib_data calibration_data`
- `T_BMP180_measurement_data temperature_value`
- `T_BMP180_measurement_data pressure_value`
- `int32_t B5_mem`

### 4.1.1 Detailed Description

BMP180 sensor class definition.

This class manages BMP180 driver.

Definition at line 56 of file Bmp180.h.

### 4.1.2 Constructor & Destructor Documentation

#### 4.1.2.1 Bmp180()

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

`Bmp180` class constructor.

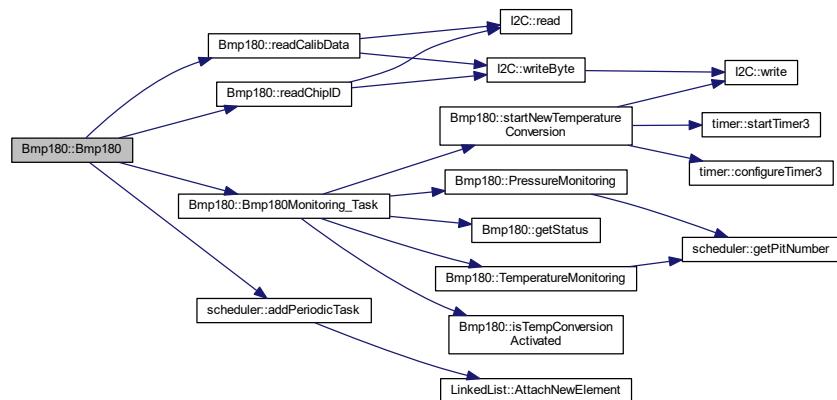
This function initializes the class `Bmp180`. It reads the chip ID and reads calibration data. If the chip ID or calibration data are incorrect, the status is set to communication failed. It also starts the periodic monitoring of the driver.

Returns

Nothing

Definition at line 21 of file Bmp180.cpp.

Here is the call graph for this function:



### 4.1.3 Member Function Documentation

#### 4.1.3.1 ActivatePressureConversion()

```
void Bmp180::ActivatePressureConversion (
    uint16_t req_period )
```

Pressure conversion activation function.

This function activates or the periodic start of pressure conversion. The requested period is transmitted in parameter.

##### Parameters

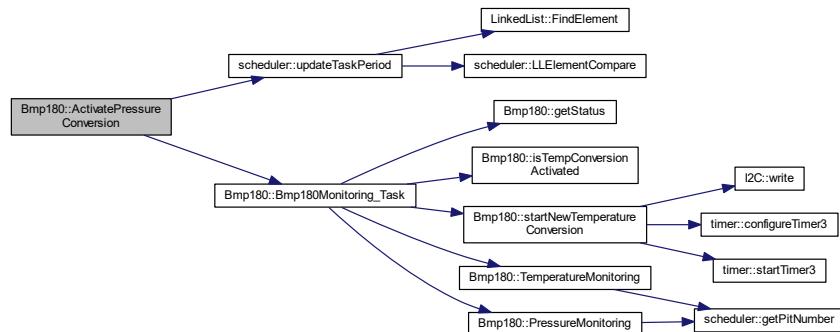
in	<i>req_period</i>	Requested period for pressure conversion
----	-------------------	--

##### Returns

Nothing

Definition at line 338 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.3.2 ActivateTemperatureConversion()

```
void Bmp180::ActivateTemperatureConversion (
    uint16_t req_period )
```

Temperature conversion activation function.

This function activates or the periodic start of temperature conversion. The requested period is transmitted in parameter.

#### Parameters

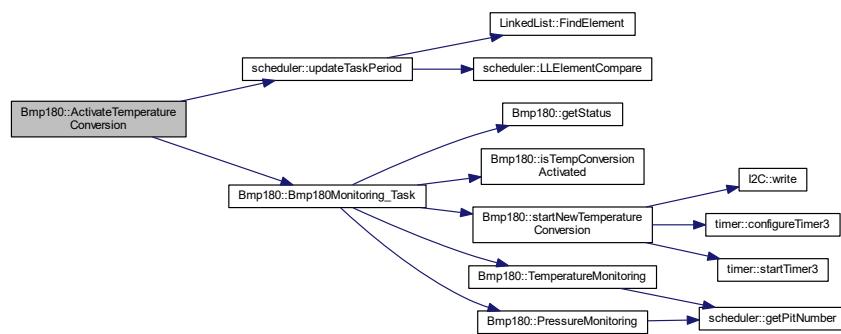
in	<i>req_period</i>	Requested period for temperature conversion
----	-------------------	---

#### Returns

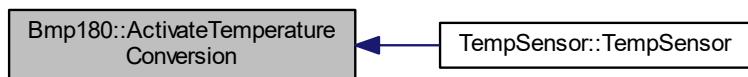
Nothing

Definition at line 328 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.3.3 Bmp180Monitoring\_Task()

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

BMP180 periodic monitoring function.

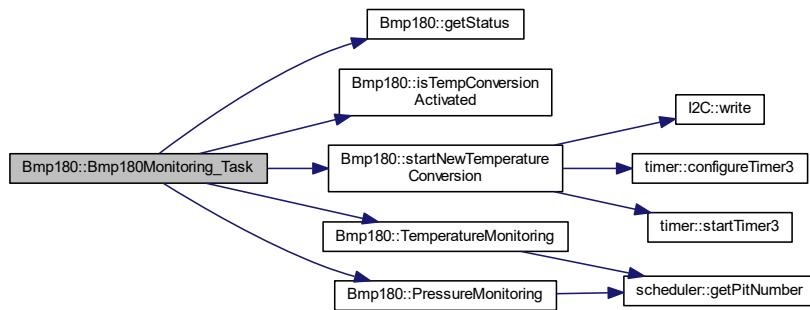
This function is in charge of monitoring the BMP180 sensor. It starts new conversions of temperature and pressure values. The result of this conversion will be retrieved by an interrupt after 4.5 ms. Temperature and pressure values time stamps are monitored and availability of measures are updated. It also monitors the status of the driver, if the driver is failed, it tries to restart the sensor device.

**Returns**

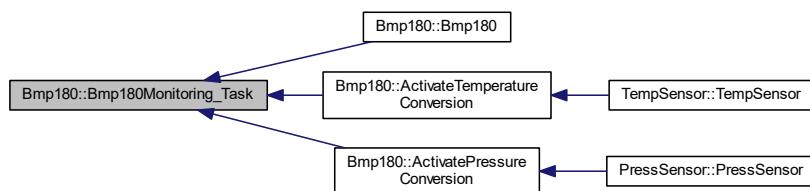
Nothing

Definition at line 305 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.3.4 CalculatePressure()

```
void Bmp180::CalculatePressure (
    uint32_t UP ) [private]
```

Pressure calculation function.

This function calculates the true pressure from the raw value from sensor according to the BMP180 datasheet. The true pressure value is stored in pressure\_value structure.

**Parameters**

in	<i>UP</i>	Raw pressure value
----	-----------	--------------------

**Returns**

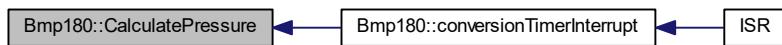
Nothing

Definition at line 275 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.1.3.5 CalculateTemperature()**

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

Temperature calculation function.

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

**Parameters**

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

**Returns**

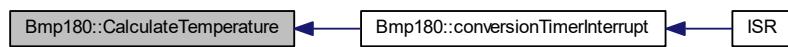
Nothing

Definition at line 264 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.3.6 conversionTimerInterrupt()

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

End of conversion interrupt.

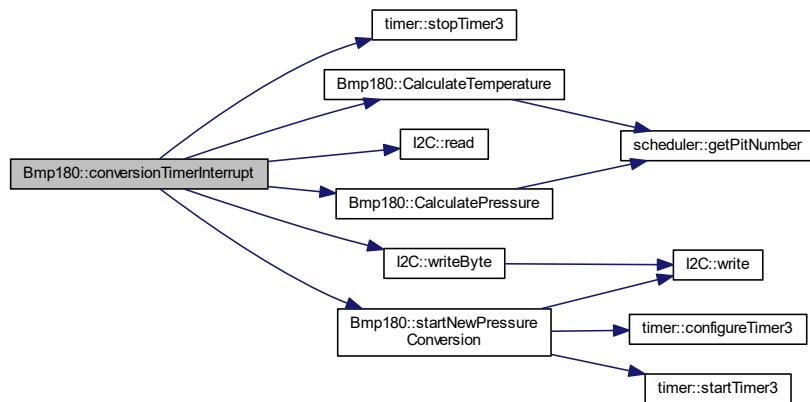
This function is called by the timer interrupt at the end of the conversion. It will retrieve the raw temperature or pressure value according to the conversion type and then calculate true value.

**Returns**

Nothing

Definition at line 207 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.3.7 getMonitoringTaskPeriod()

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

Task period get function.

This function returns the task period of the monitoring function.

##### Returns

Task period

Definition at line 127 of file Bmp180.h.

#### 4.1.3.8 getPressureValue()

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

Pressure reading function.

This function is used to read the pressure value from BMP180 sensor.

##### Parameters

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

##### Returns

True if a pressure measurement is available, false otherwise.

Definition at line 158 of file Bmp180.cpp.

---

Here is the caller graph for this function:



#### 4.1.3.9 getStatus()

`T_BMP180_Status Bmp180::getStatus () [inline]`

Driver status get function.

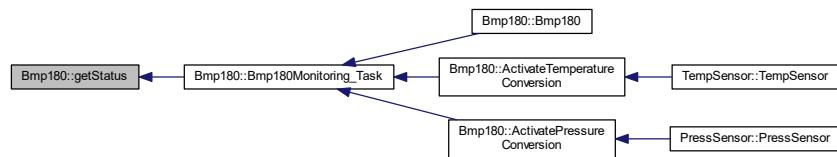
This function returns the status of the driver.

##### Returns

Driver status

Definition at line 116 of file Bmp180.h.

Here is the caller graph for this function:



#### 4.1.3.10 getTemperatureValue()

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

Temperature reading function.

This function is used to read the temperature value from BMP180 sensor.

**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 151 of file Bmp180.cpp.

Here is the caller graph for this function:

**4.1.3.11 isPressConversionActivated()**

```
bool Bmp180::isPressConversionActivated ( ) [inline]
```

Pressure conversion activation flag get function.

This function returns the activation status of the pressure conversion.

**Returns**

True if pressure conversion is activated, false otherwise.

Definition at line 221 of file Bmp180.h.

**4.1.3.12 isTempConversionActivated()**

```
bool Bmp180::isTempConversionActivated ( ) [inline]
```

Temperature conversion activation flag get function.

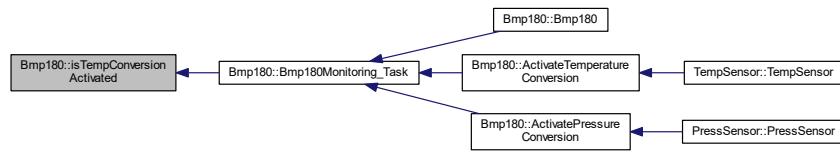
This function returns the activation status of the temperature conversion.

**Returns**

True if temperature conversion is activated, false otherwise.

Definition at line 210 of file Bmp180.h.

Here is the caller graph for this function:

**4.1.3.13 PressureMonitoring()**

```
void Bmp180::PressureMonitoring ( )
```

Pressure value monitoring function.

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

**Returns**

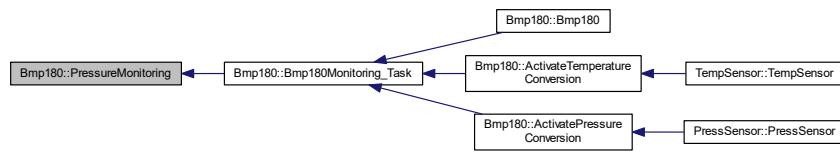
Nothing

Definition at line 322 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.3.14 readCalibData()

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

Calibration data reading function.

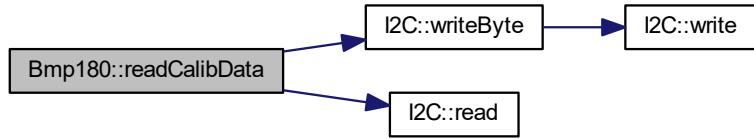
This function reads the pressure and temperature calibration data in BMP180 EEPROM using I<sub>2</sub>C bus. It also updates the driver status if the communication with the device is failed.

##### Returns

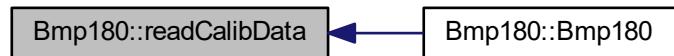
Nothing.

Definition at line 63 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.3.15 readChipID()

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

Chip ID read function.

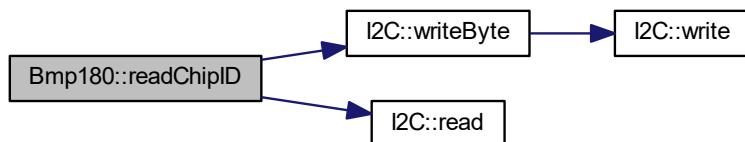
This function reads the ID of the sensor chip using I<sub>2</sub>C bus. It also updates the driver status if the communication is failed.

**Returns**

Nothing

Definition at line 132 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.3.16 startNewPressureConversion()

```
void Bmp180::startNewPressureConversion( )
```

Starts a new pressure conversion.

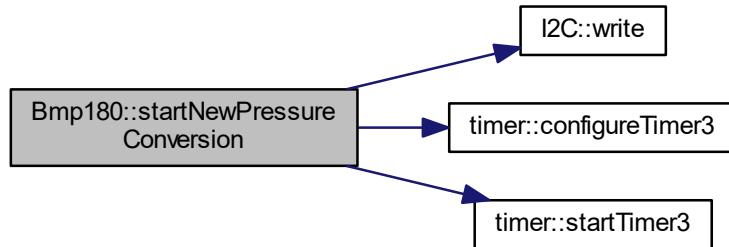
This function starts a new pressure conversion by writing 0x74 into register 0xF4 of sensor. It also starts a timer to retrieve sensor data after the conversion time.

Returns

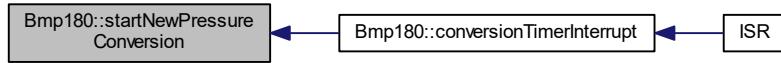
Nothing

Definition at line 186 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.3.17 `startNewTemperatureConversion()`

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

Starts a new temperature conversion.

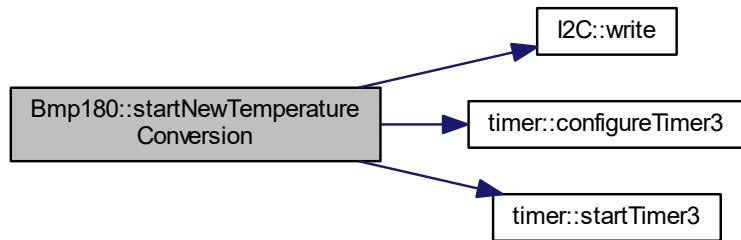
This function starts a new temperature conversion by writing 0x2E into register 0xF4 of sensor. It also starts a timer to retrieve sensor data after the conversion time.

**Returns**

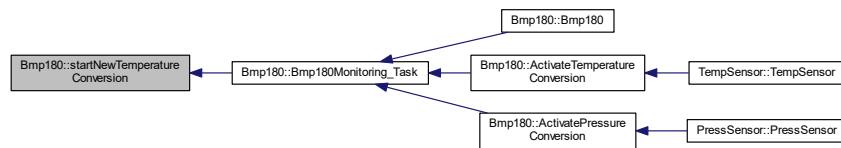
Nothing

Definition at line 165 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.1.3.18 StopPressureConversion()**

```
void Bmp180::StopPressureConversion( )
```

Pressure conversion stop function.

This function stops or the periodic start of pressure conversion.

**Returns**

Nothing

Definition at line 353 of file Bmp180.cpp.

#### 4.1.3.19 StopTemperatureConversion()

```
void Bmp180::StopTemperatureConversion ( )
```

Temperature conversion stop function.

This function stops or the periodic start of temperature conversion.

##### Returns

Nothing

Definition at line 348 of file Bmp180.cpp.

#### 4.1.3.20 TemperatureMonitoring()

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

Temperature value monitoring function.

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

##### Returns

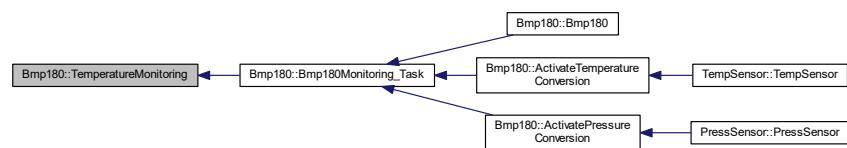
Nothing

Definition at line 316 of file Bmp180.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.4 Member Data Documentation

##### 4.1.4.1 B5\_mem

```
int32_t Bmp180::B5_mem [private]
```

Memorization of B5 coefficient (computed by temperature formula and used for pressure)

Definition at line 269 of file Bmp180.h.

##### 4.1.4.2 calibration\_data

```
T_BMP180_calib_data Bmp180::calibration_data [private]
```

Calibration data of the sensor

Definition at line 253 of file Bmp180.h.

##### 4.1.4.3 chip\_id

```
uint8_t Bmp180::chip_id [private]
```

Sensor chip ID

Definition at line 228 of file Bmp180.h.

##### 4.1.4.4 i2c\_drv\_ptr

```
I2C* Bmp180::i2c_drv_ptr [private]
```

Pointer to the [I2C](#) driver object

Definition at line 227 of file Bmp180.h.

##### 4.1.4.5 isPressConvActivated

```
bool Bmp180::isPressConvActivated [private]
```

Pressure conversion activation flag

Definition at line 232 of file Bmp180.h.

#### 4.1.4.6 `isTempConvActivated`

```
bool Bmp180::isTempConvActivated [private]
```

Temperature conversion activation flag

Definition at line 231 of file Bmp180.h.

#### 4.1.4.7 `pressure_value`

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

Pressure data structure

Definition at line 267 of file Bmp180.h.

#### 4.1.4.8 `status`

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

Sensor status

Definition at line 229 of file Bmp180.h.

#### 4.1.4.9 `task_period`

```
uint16_t Bmp180::task_period [private]
```

Period of the monitoring task

Definition at line 230 of file Bmp180.h.

#### 4.1.4.10 `temperature_value`

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

Temperature data structure

Definition at line 266 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**

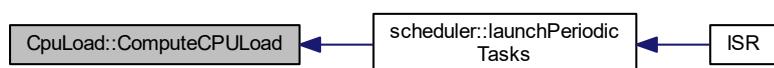
Nothing

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 getCurrentCPULoad()

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

Get current CPU load value.

This function returns the current CPU load value

##### Returns

Current CPU load value

Definition at line 47 of file CpuLoad.h.

Here is the caller graph for this function:



#### 4.2.3.4 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:

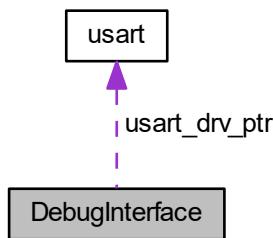
- [DebugManagement.h](#)

## 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.

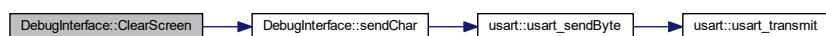
This function clears the entire display by sending the \f character on the USART line.

**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.

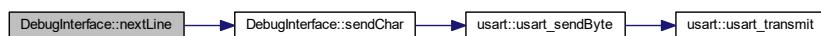
This function goes to the next line on the console display. It sends the two characters `\n` and `\r` on the USART line.

##### 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.

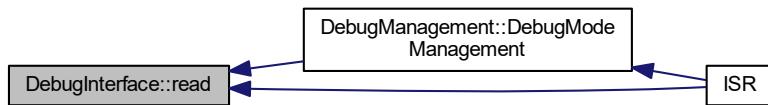
This function will read the last received byte on USART link

##### 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

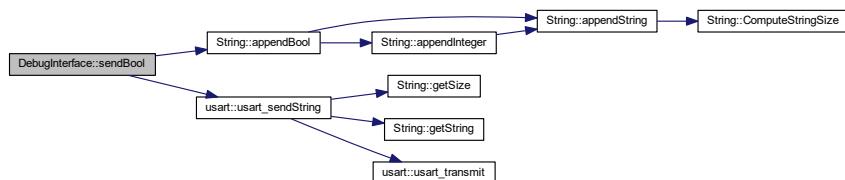
in	<code>data</code>	boolean data to be sent
in	<code>isText</code>	<a href="#">String</a> conversion configuration

#### Returns

Nothing

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

Here is the call graph for this function:



#### 4.4.3.5 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**

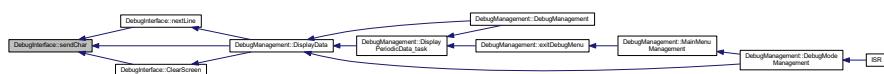
Nothing

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**

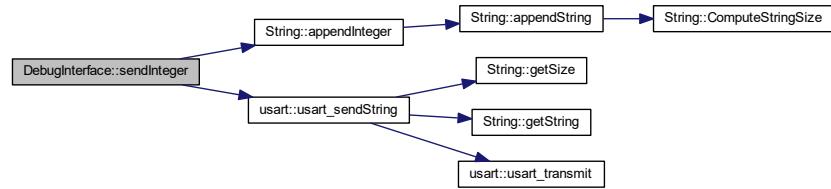
<b>in</b>	<b>data</b>	integer data to be sent
<b>in</b>	<b>base</b>	numerical base used to convert integer into string (between 2 and 36)

**Returns**

Nothing

Definition at line 49 of file DebugInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.4.3.7 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

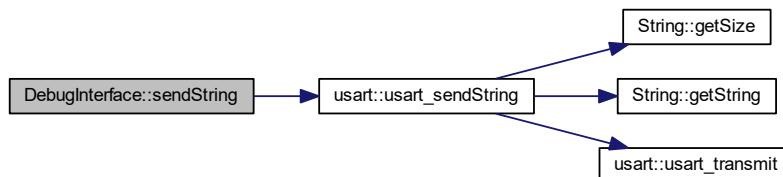
in	<i>str</i>	Pointer to the string being sent
----	------------	----------------------------------

##### Returns

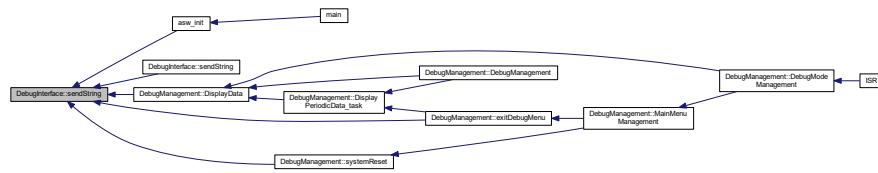
Nothing

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

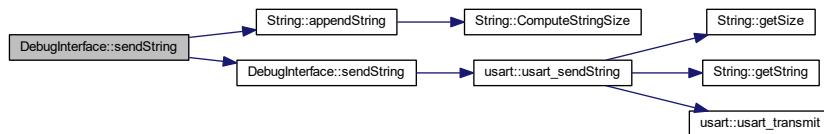
in	<i>str</i>	Pointer to the chain to send.
----	------------	-------------------------------

##### 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

```
uart* 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:

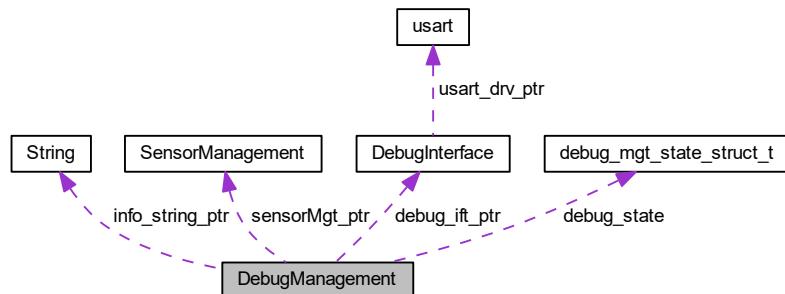
- [DebugInterface.h](#)
- [DebugInterface.cpp](#)

## 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.

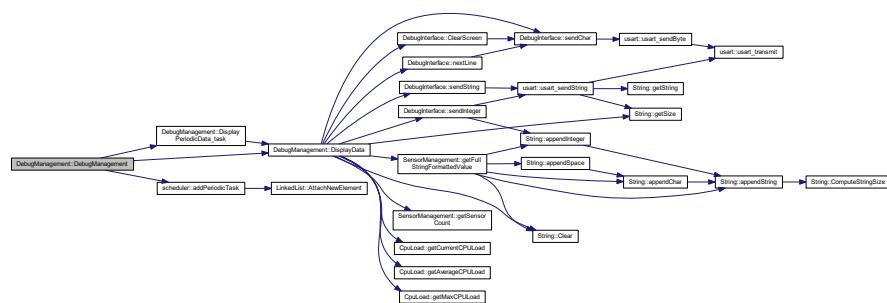
This function initializes the class. If needed, it creates a new instance of debug interface object.

#### 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 :

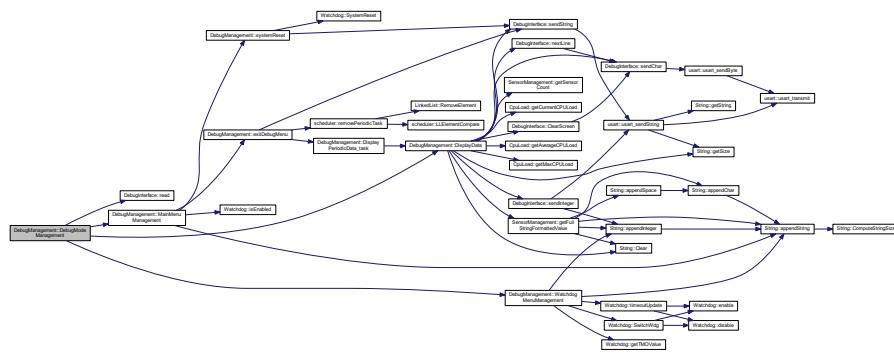
- MAIN\_MENU state : handles user choice in main menu and selects next state
- WDG\_MENU state : handles user choice in watchdog menu and selects next state  
It is called each time a data is received on USART and debug mode is active.

## Returns

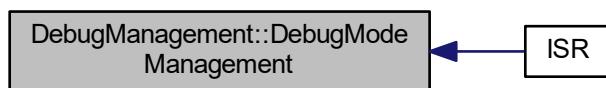
True if the debug mode shall be closed, false otherwise

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.

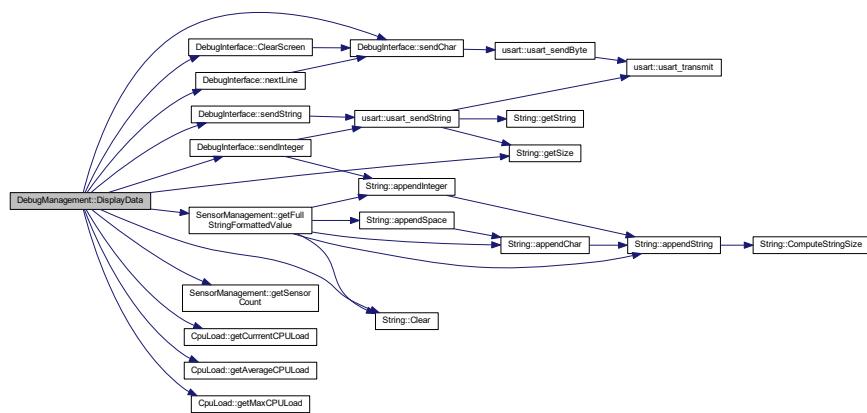
This task displays the menu and periodic data (temperature, humidity and CPU load) on USART screen.

**Returns**

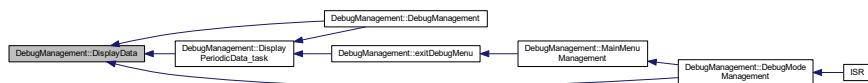
Nothing

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 usart link.

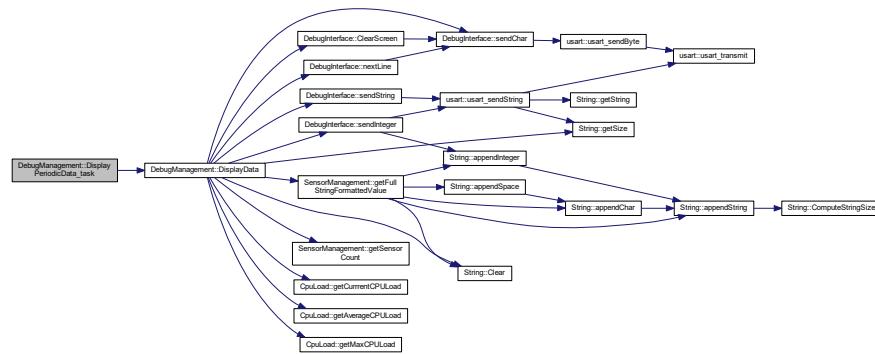
This task displays the menu and periodic data (temperature, humidity and CPU load) on usart screen. It only calls the function `DisplayData`.

**Returns**

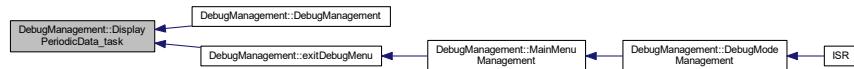
Nothing

Definition at line 195 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.5.3.4 exitDebugMenu()**

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

Debug menu exit function.

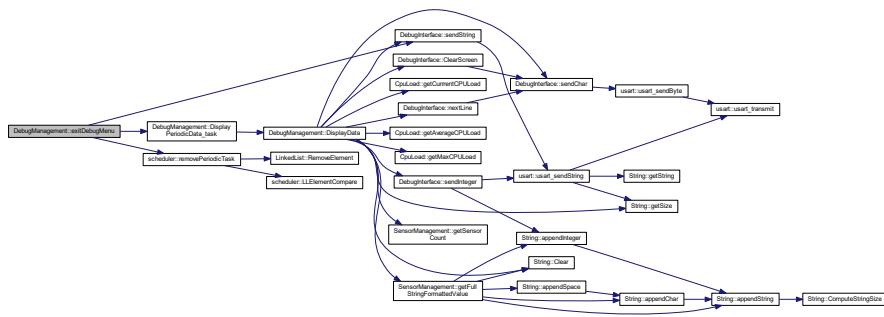
This function prepares the exit of debug menu. It writes the message "Bye !" on the screen and removes the periodic task from the scheduler.

**Returns**

Nothing.

Definition at line 230 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.5.3.5 getIfpt()

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

Interface pointer get function.

This function returns the pointer to the debug interface object

##### Returns

Pointer to debug interface

Definition at line 95 of file DebugManagement.h.

#### 4.5.3.6 getInfoStringPtr()

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

Info string get function.

This function returns the pointer to the info string to display

##### Returns

Info string pointer

Definition at line 115 of file DebugManagement.h.

#### 4.5.3.7 getMenuStringPtr()

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

Menu string get function.

This function returns the pointer to the menu string to display

##### Returns

Menu string pointer

Definition at line 105 of file DebugManagement.h.

#### 4.5.3.8 MainMenuManagement()

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

Main menu management.

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

##### Parameters

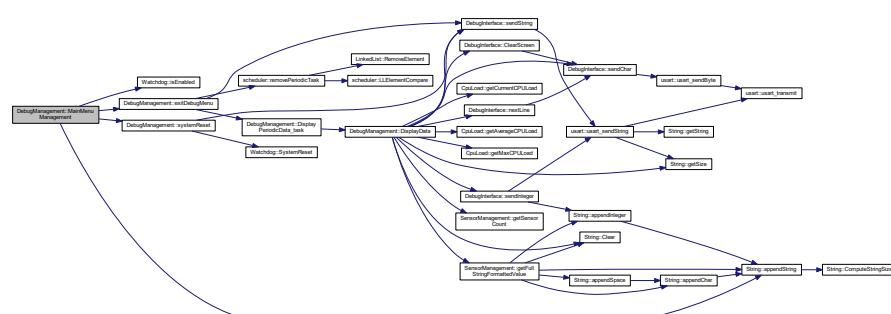
in	<i>rcv_char</i>	Character received on USART bus.
----	-----------------	----------------------------------

##### Returns

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

Definition at line 347 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.5.3.9 setInfoStringPtr()

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

Info message setting function.

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

##### Parameters

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

##### Returns

Nothing

Definition at line 126 of file DebugManagement.h.

#### 4.5.3.10 systemReset()

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

System reset function.

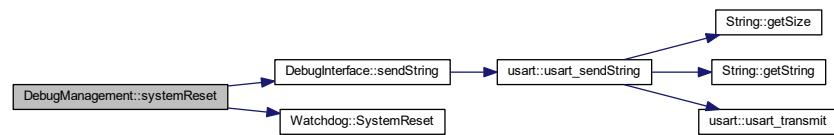
This function provokes a reset of the system. It displays a message on the screen and calls the reset function from watchdog class.

**Returns**

Nothing.

Definition at line 236 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.5.3.11 WatchdogMenuManagement()

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

[Watchdog](#) menu management function.

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

**Parameters**

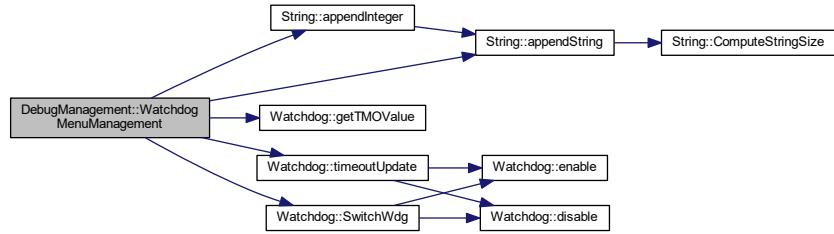
in	<code>rcv_char</code>	Character received on USART bus.
----	-----------------------	----------------------------------

**Returns**

Nothing.

Definition at line 242 of file DebugManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.5.4 Member Data Documentation

##### 4.5.4.1 debug\_ift\_ptr

```
DebugInterface* DebugManagement::debug_ift_ptr [private]
```

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

Definition at line 133 of file DebugManagement.h.

##### 4.5.4.2 debug\_state

```
debug_mgt_state_struct_t DebugManagement::debug_state [private]
```

Structure containing debug states for each menu

Definition at line 137 of file DebugManagement.h.

#### 4.5.4.3 info\_string\_ptr

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

Pointer to the info message to display

Definition at line 136 of file DebugManagement.h.

#### 4.5.4.4 isInfoStringDisplayed

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

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

Definition at line 138 of file DebugManagement.h.

#### 4.5.4.5 menu\_string\_ptr

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

Pointer to the current menu string to display

Definition at line 135 of file DebugManagement.h.

#### 4.5.4.6 sensorMgt\_ptr

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

Pointer to the sensor management object

Definition at line 134 of file DebugManagement.h.

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

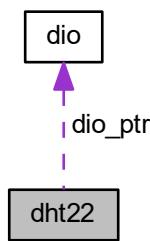
- [DebugManagement.h](#)
- [DebugManagement.cpp](#)

## 4.6 dht22 Class Reference

DHT 22 driver class.

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

Collaboration diagram for dht22:



### Public Member Functions

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

### Private Member Functions

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

### Private Attributes

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

#### 4.6.1 Detailed Description

DHT 22 driver class.

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

Definition at line 19 of file dht22.h.

#### 4.6.2 Constructor & Destructor Documentation

##### 4.6.2.1 dht22()

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

[dht22](#) class constructor.

Initializes the class [dht22](#).

##### Parameters

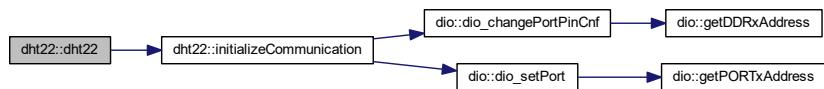
in	<i>port</i>	Encoded configuration of the port used for 1-wire communication.
----	-------------	--

##### Returns

Nothing

Definition at line 24 of file dht22.cpp.

Here is the call graph for this function:



#### 4.6.3 Member Function Documentation

#### 4.6.3.1 getHumidity()

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

Humidity get function.

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

### Parameters

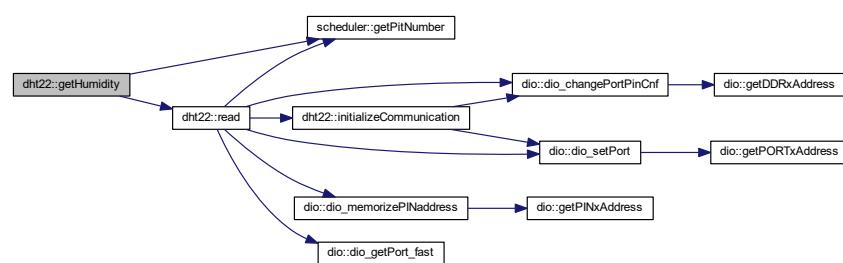
in	<i>humidity</i>	Address where the humidity shall be written
----	-----------------	---

### Returns

Validity of the data

Definition at line 220 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 4.6.3.2 getTemperature()

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

Temperature get function.

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

### Parameters

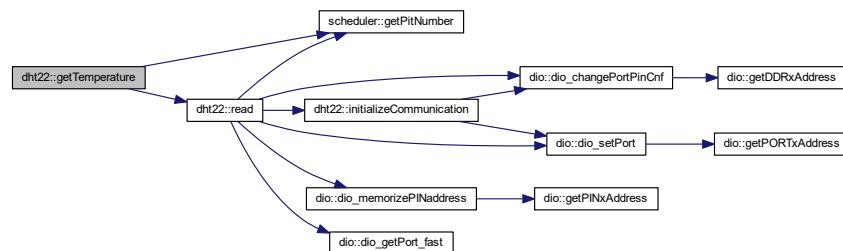
in	<i>temperature</i>	Address where the temperature shall be written
----	--------------------	--

**Returns**

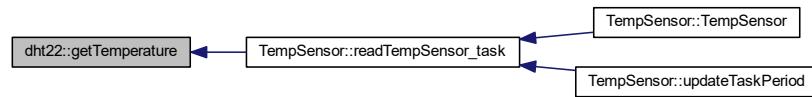
Validity of the data

Definition at line 231 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.6.3.3 initializeCommunication()**

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

Initializes the communication.

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

**Returns**

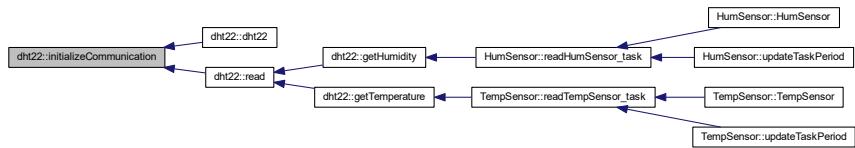
Nothing

Definition at line 210 of file dht22.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.6.3.4 `read()`

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

Reads the data from DHT22.

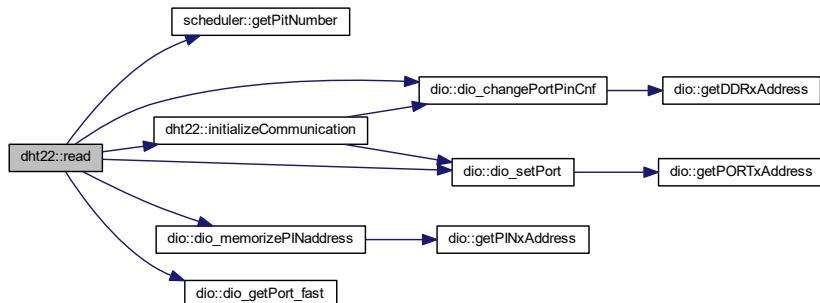
This function communicates with DHT22 using 1-wire protocol to read raw values of temperature and humidity. A checksum check is done when communication is finished to validate the received data. Validity of the data, temperature and humidity values and memorized in the associated class members.

##### Returns

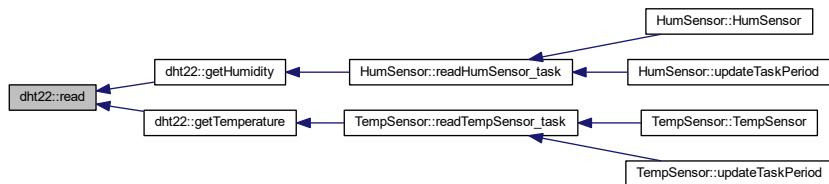
Nothing

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

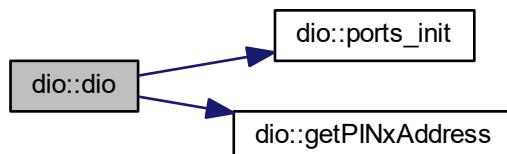
Initializes class dio and calls DIO hardware initialization function

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

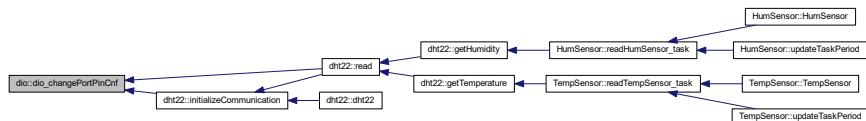
Nothing

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

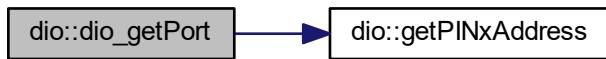
in	<i>portcode</i>	Encoded pin and register index
----	-----------------	--------------------------------

**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.

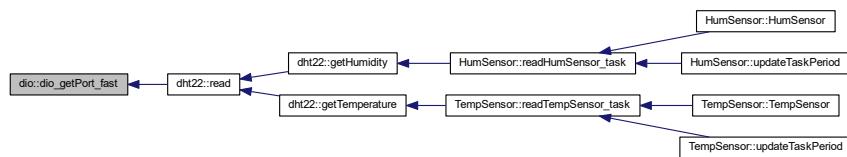
This function gets the logical value of the memorized pin. The corresponding port and pin index are stored in class members PINx\_addr\_mem and PINx\_idx\_mem. This mechanism is used to speed up reading time as this function no longer needs to extract register address and pin index from portcode.

**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

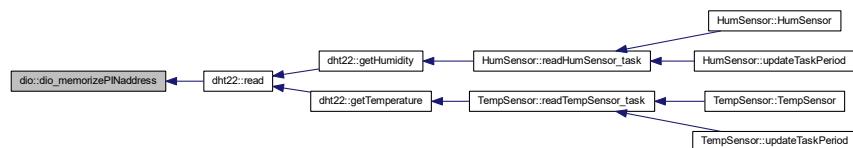
Nothing

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**

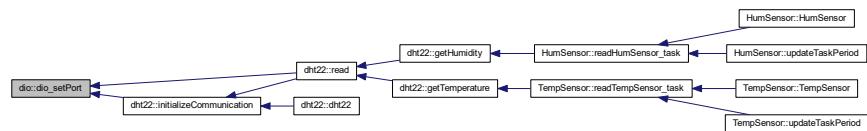
Nothing

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**

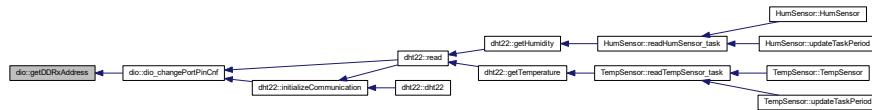
in	<i>portcode</i>	Encoded port code
----	-----------------	-------------------

**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

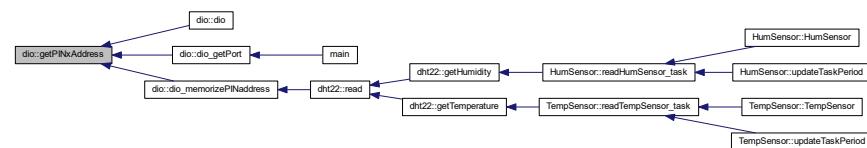
in	<i>portcode</i>	Encoded port code
----	-----------------	-------------------

##### 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**

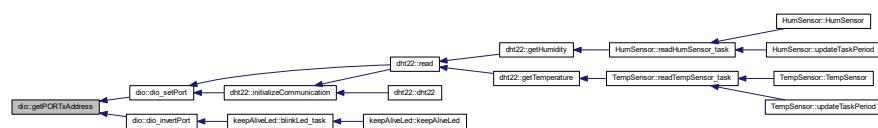
in	<i>portcode</i>	Encoded port code
----	-----------------	-------------------

**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.

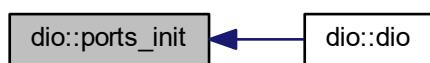
This function initializes digital ports as input or output and sets their initial values

**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:

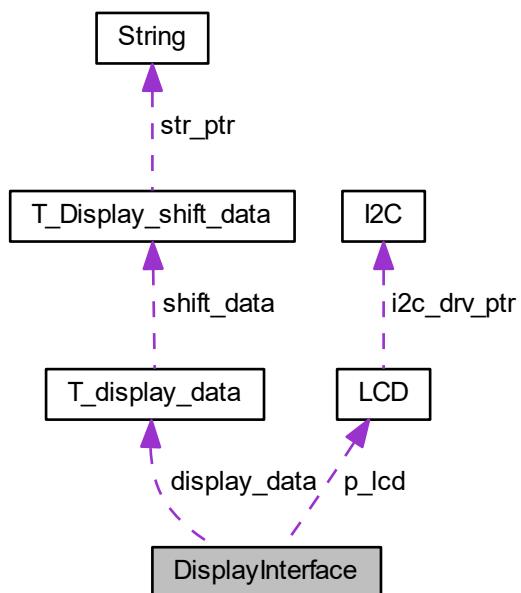
- [dio.h](#)
- [dio.cpp](#)

## 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

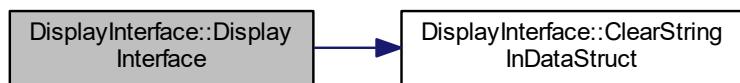
in	<i>LCD_init_cnf</i>	Initial configuration of the screen
----	---------------------	-------------------------------------

#### 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.

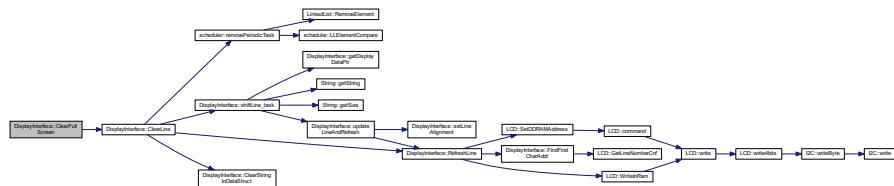
This function clears the entire display. It uses the ClearLine function on every line of the screen.

## 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

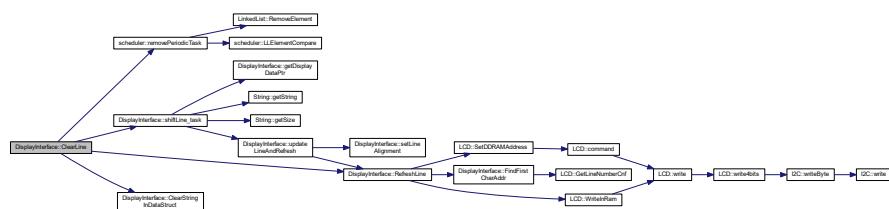
in	<i>line</i>	Line to clear
----	-------------	---------------

**Returns**

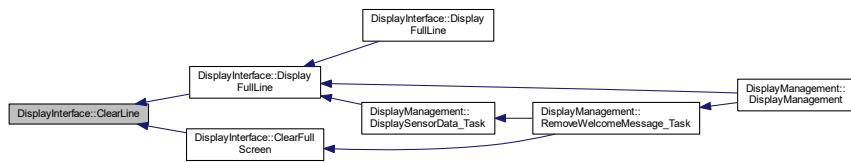
True if the line has been cleared, false otherwise

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**

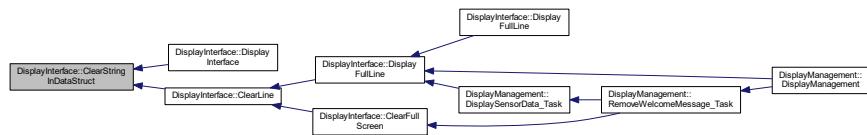
in	line	Line to clear
----	------	---------------

**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

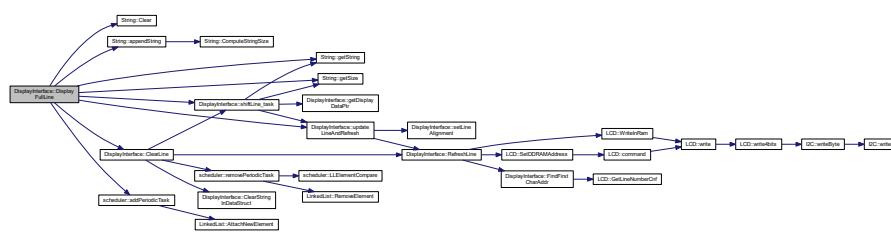
in	<i>str</i>	Pointer to the string to display
in	<i>size</i>	Size of the string to display
in	<i>line</i>	Index of the line where the string shall be displayed
in	<i>mode</i>	Display mode
in	<i>alignment</i>	Requested alignment for the line

## Returns

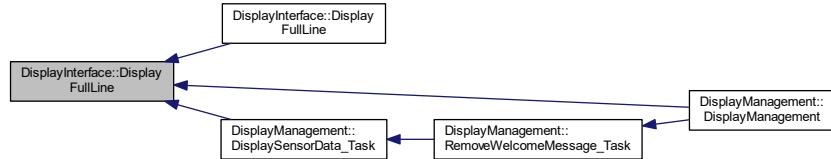
True if the line has been correctly displayed, false otherwise

Definition at line 59 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



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

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

Line display function.

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

##### Parameters

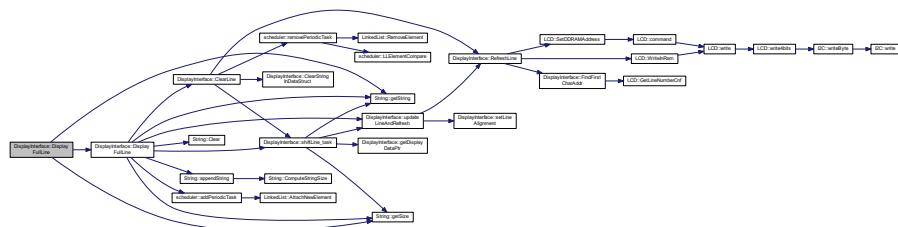
in	<i>str</i>	Pointer to the string to display
in	<i>line</i>	Index of the line where the string shall be displayed
in	<i>mode</i>	Display mode
in	<i>alignment</i>	Requested alignment for the line

##### Returns

True if the line has been correctly displayed, false otherwise

Definition at line 143 of file DisplayInterface.cpp.

Here is the call graph for this function:



#### 4.8.3.6 FindFirstCharAddr()

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

Finds start address of a line.

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

##### Parameters

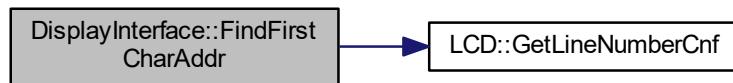
<code>in</code>	<code>line</code>	Line which address shall be found
-----------------	-------------------	-----------------------------------

##### Returns

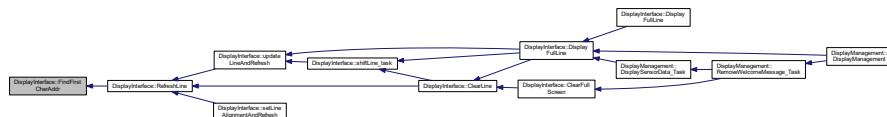
Address in DDRAM of the first character of the line

Definition at line 185 of file `DisplayInterface.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.8.3.7 getDisplayDataPtr()

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

Display data get function.

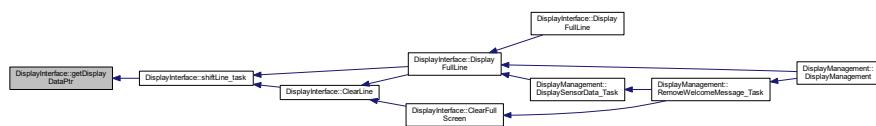
This function returns a pointer to the display data structure.

**Returns**

Pointer to display data structure.

Definition at line 154 of file DisplayInterface.h.

Here is the caller graph for this function:

**4.8.3.8 IsLineEmpty()**

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

Empty line get function.

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

**Parameters**

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

**Returns**

True if the line is empty, false otherwise

Definition at line 277 of file DisplayInterface.cpp.

**4.8.3.9 RefreshLine()**

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

Line refresh function.

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

**Parameters**

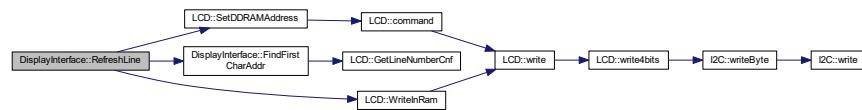
in	<i>line</i>	Line to refresh
----	-------------	-----------------

**Returns**

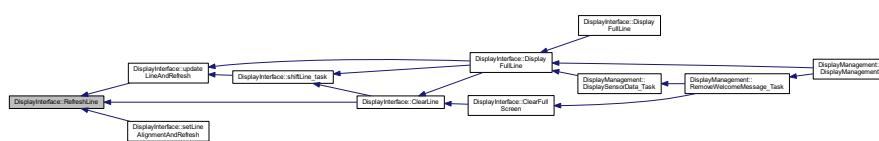
Nothing

Definition at line 164 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.8.3.10 setLineAlignment()**

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

Text alignment setting function.

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

**Parameters**

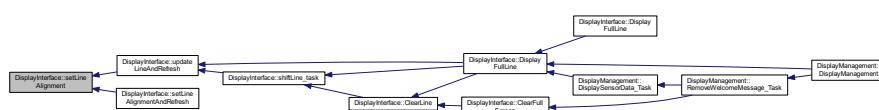
in	<i>line</i>	Line to update
----	-------------	----------------

**Returns**

Nothing

Definition at line 334 of file DisplayInterface.cpp.

Here is the caller graph for this function:



#### 4.8.3.11 setLineAlignmentAndRefresh()

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

Text alignment function.

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

#### Parameters

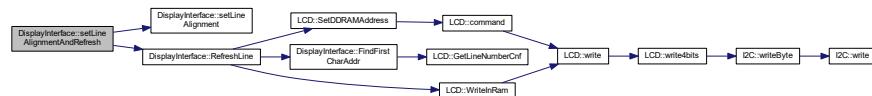
in	<i>line</i>	Requested line to update
in	<i>alignment</i>	Requested alignment for the text

#### Returns

Nothing

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

Here is the call graph for this function:



#### 4.8.3.12 shiftLine\_task()

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

Line shifting periodic task.

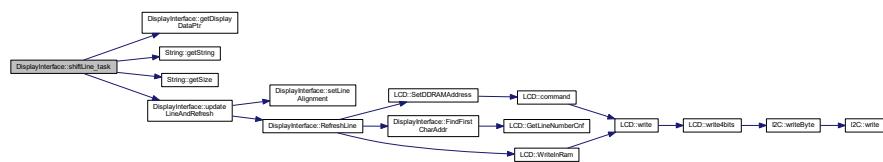
This function is called periodically by the scheduler. It shifts all the lines in line shifting mode and updates the data structures.

**Returns**

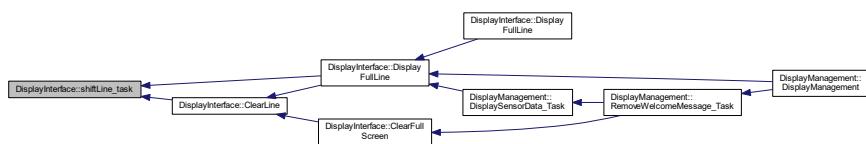
Nothing

Definition at line 286 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.8.3.13 updateLineAndRefresh()**

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

Line data string update function.

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

**Parameters**

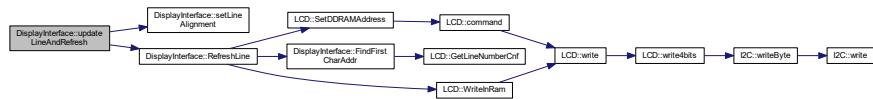
<b>in</b>	<b>str</b>	Pointer to the string to display
<b>in</b>	<b>size</b>	Size of the string
<b>in</b>	<b>line</b>	Line to update

**Returns**

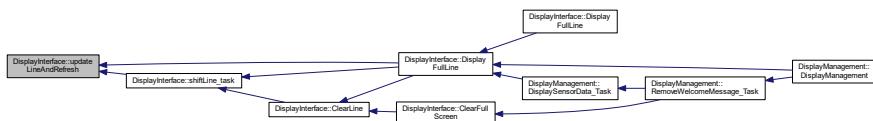
Nothing

Definition at line 148 of file DisplayInterface.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.8.4 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:

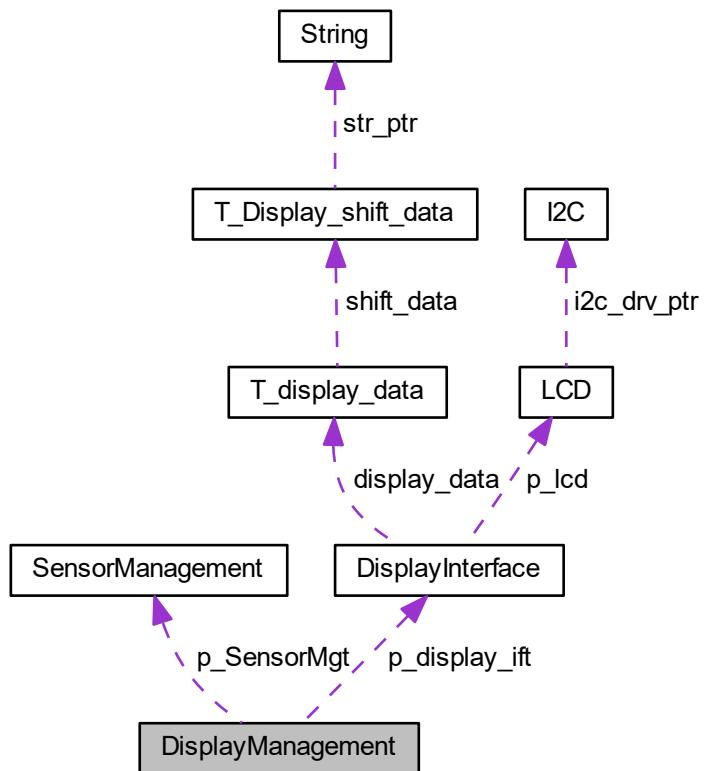
- [DisplayInterface.h](#)
- [DisplayInterface.cpp](#)

## 4.9 DisplayManagement Class Reference

Display management class.

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

Collaboration diagram for DisplayManagement:



## Public Member Functions

- [DisplayManagement \(\)](#)  
*Class constructor.*
- [DisplayInterface \\* GetIfpPointer \(\)](#)  
*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.

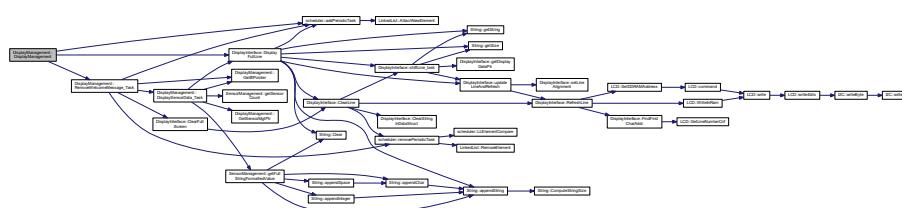
It creates a display interface object and initializes all class variables.

#### Returns

Nothing

Definition at line 30 of file [DisplayManagement.cpp](#).

Here is the call graph for this function:



### 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.

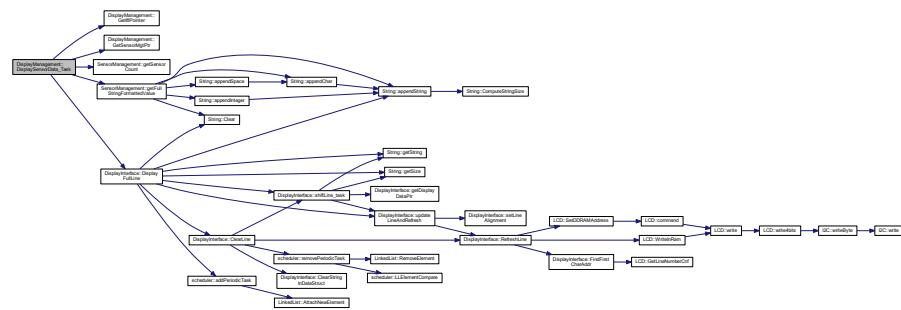
It is called periodically by scheduler.

#### 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 GetIfpPointer()

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

Interface pointer get function.

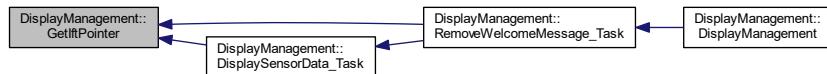
This function returns the pointer to the display interface object

##### 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.

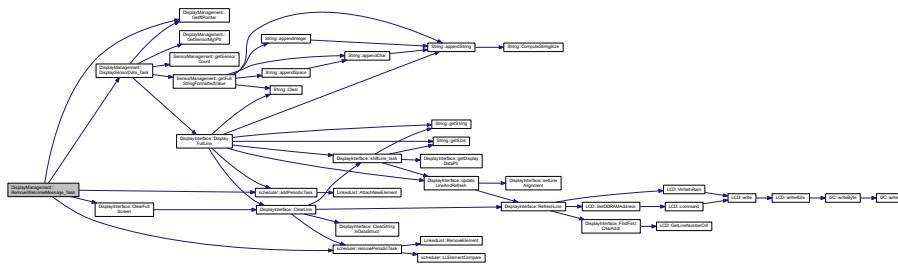
This task clears the welcome message from the screen and start periodic display of sensor data. This task shall be added in scheduler when the welcome message is displayed on screen. As it shall be called only once, the task removes itself from the scheduler after the first call.

## 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:

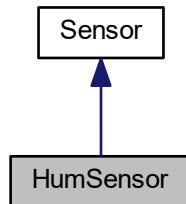
- [DisplayManagement.h](#)
- [DisplayManagement.cpp](#)

## 4.10 HumSensor Class Reference

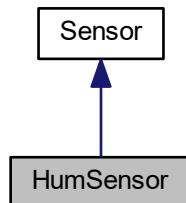
Class for humidity sensor.

```
#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.

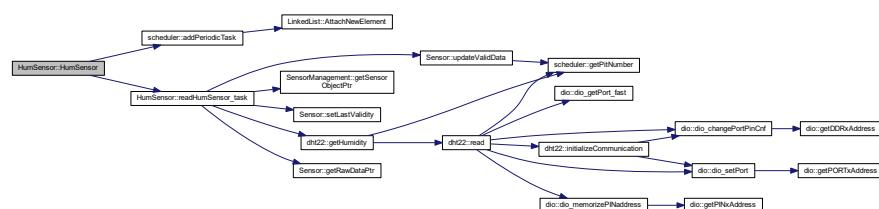
This function initializes all data of the class [HumSensor](#). If needed, it creates a new instance of the DHT22 sensor object. It also adds the periodic task in the scheduler.

#### 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.

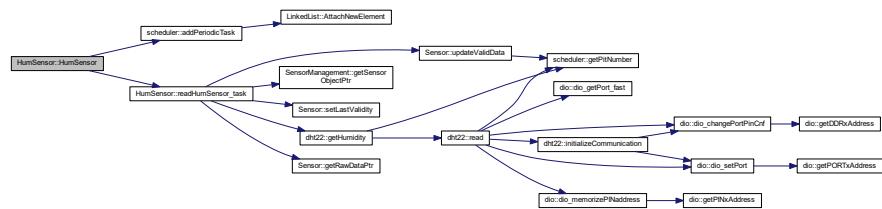
This function initializes all data of the class `HumSensor`. It sets validity timeout and task period to the given value. If needed, it creates a new instance of the DHT22 sensor object. It also adds the periodic task in the scheduler.

## 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.

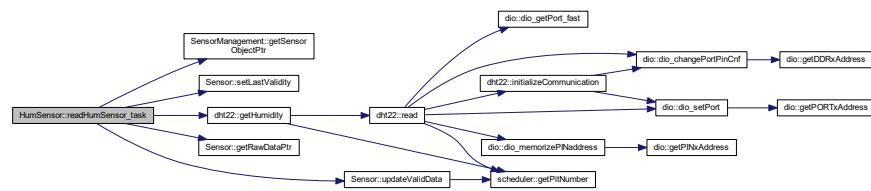
This task reads humidity data using DHT22 driver. It is called periodically.

**Returns**

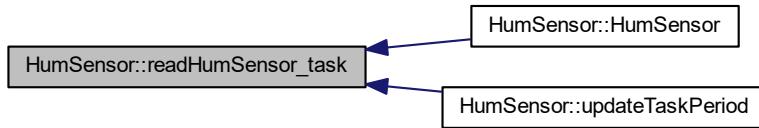
Nothing

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**

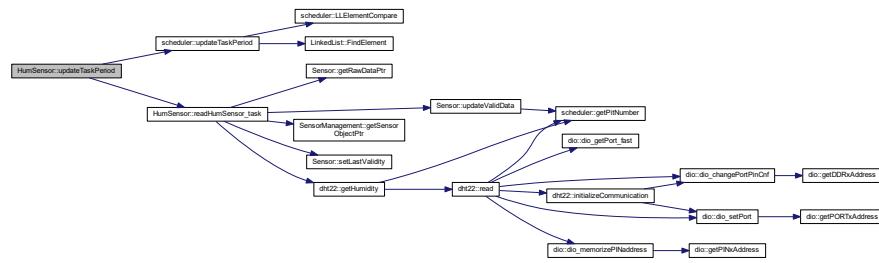
in	<i>period</i>	New period of the task
----	---------------	------------------------

**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 <a href="#">I2C</a> bus (in Hz)
----	------------------	---

##### Returns

Nothing

Definition at line 17 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 141 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

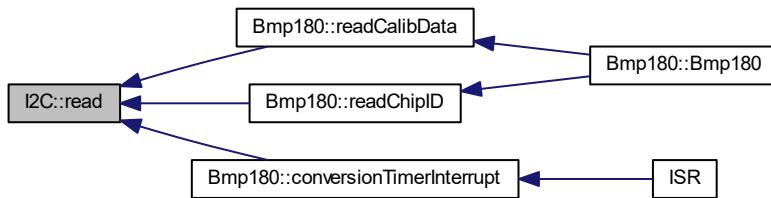
in	<i>i2c_address</i>	I2C address of the device
in	<i>size</i>	Number of bytes to read
out	<i>buf_ptr</i>	Pointer to the start of the reception buffer.

**Returns**

True if the receive process has succeeded, false otherwise

Definition at line 81 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 136 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

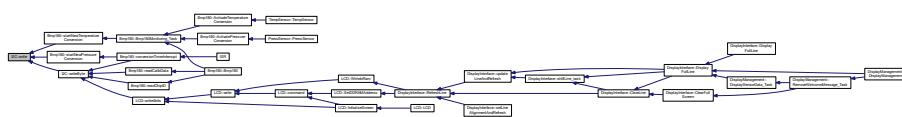
in	<i>data</i>	Pointer to the data to send
in	<i>tx_address</i>	I <sup>2</sup> C address of the device
in	<i>size</i>	Number of bytes to send
in	<i>sendStopCond</i>	Defines if the stop condition shall be sent or not

## Returns

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

Definition at line 29 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

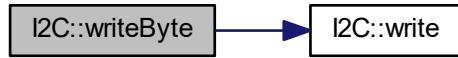
in	<i>data</i>	Data to send
in	<i>tx_address</i>	I <sup>2</sup> C address of the device
in	<i>sendStopCond</i>	Defines if the stop condition shall be sent or not

## Returns

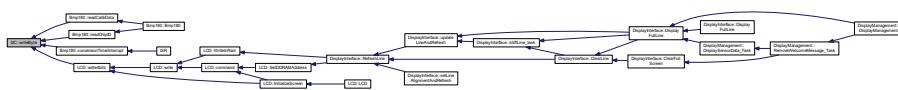
True if transmission is completed, False if an error has occurred

Definition at line 24 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.

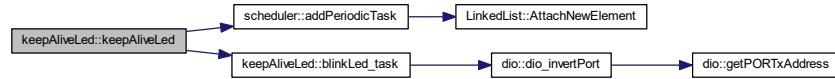
This function initializes the class `keepAliveLed`

#### 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.

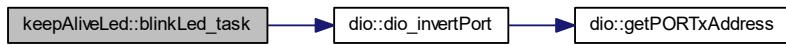
This function is inserted into the scheduler. It changes the state of the LED output to make it blink

##### 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:

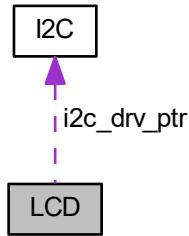
- [keepAliveLed.h](#)
- [keepAliveLed.cpp](#)

## 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

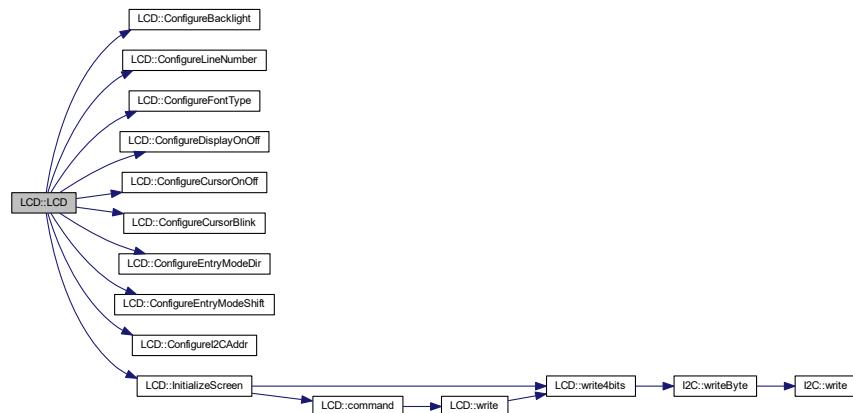
in	<code>init_conf</code>	Initial configuration structure
----	------------------------	---------------------------------

**Returns**

Nothing

Definition at line 18 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

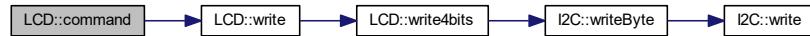
in	<i>cmd</i>	Requested command
----	------------	-------------------

**Returns**

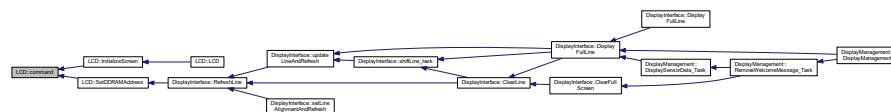
Nothing

Definition at line 122 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.13.3.2 ConfigureBacklight()

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

Backlight configuration function.

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

##### Parameters

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

##### Returns

Nothing

Definition at line 178 of file LCD.h.

Here is the caller graph for this function:



#### 4.13.3.3 ConfigureCursorBlink()

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

Cursor blinking configuration function.

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

##### Parameters

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

##### Returns

Nothing

Definition at line 238 of file LCD.h.

Here is the caller graph for this function:



#### 4.13.3.4 ConfigureCursorOnOff()

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

Cursor configuration function.

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

##### Parameters

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

##### Returns

Nothing

Definition at line 226 of file LCD.h.

Here is the caller graph for this function:



#### 4.13.3.5 ConfigureDisplayOnOff()

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

Display configuration function.

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

##### Parameters

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

##### Returns

Nothing

Definition at line 214 of file LCD.h.

Here is the caller graph for this function:



#### 4.13.3.6 ConfigureEntryModeDir()

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

Entry mode direction configuration function.

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

**Parameters**

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

**Returns**

Nothing

Definition at line 250 of file LCD.h.

Here is the caller graph for this function:



#### 4.13.3.7 ConfigureEntryModeShift()

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

Entry mode shift configuration function.

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

**Parameters**

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

**Returns**

Nothing

Definition at line 262 of file LCD.h.

Here is the caller graph for this function:



#### 4.13.3.8 ConfigureFontType()

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

Font configuration function.

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

**Parameters**

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

**Returns**

Nothing

Definition at line 202 of file LCD.h.

Here is the caller graph for this function:



#### 4.13.3.9 ConfigureI2CAddr()

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

I2C address configuration function.

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

##### Parameters

in	param	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

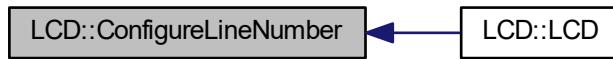
in	param	Configuration value
----	-------	---------------------

##### 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.

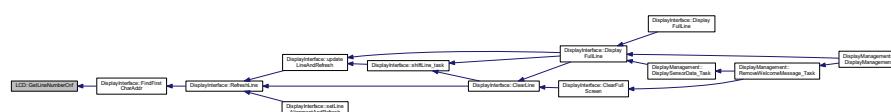
This function returns the line number configuration of the screen : 1 or 2 lines mode.

##### 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 72 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

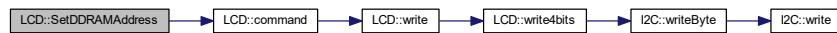
in	<code>addr</code>	New DDRAM address
----	-------------------	-------------------

**Returns**

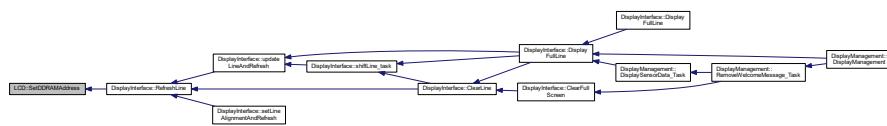
Nothing

Definition at line 165 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**

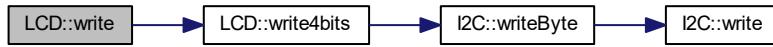
in	<i>data</i>	8-bit data for D0-7 pins of screen
in	<i>mode</i>	Requested mode for <a href="#">LCD</a> communication

**Returns**

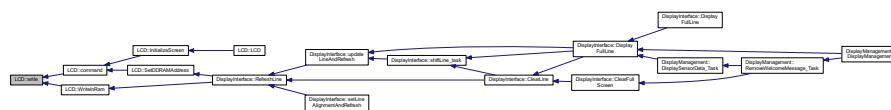
Nothing

Definition at line 62 of file LCD.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.13.3.16 write4bits()

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

I2C write function for 4-bits mode.

This function sends the requested 8-bits data on the I<sub>2</sub>C bus. The backlight pin is also set/clear according to the configuration. The function sends the data a first time with EN pin set, then a second time with EN bit clear.

## Parameters

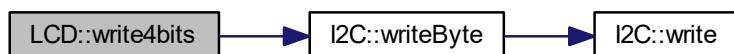
in *data* 8-bit data to send

## Returns

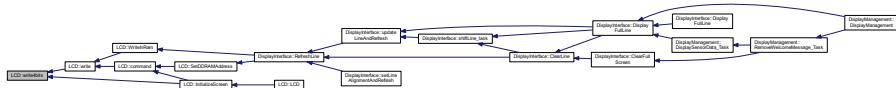
Nothing

Definition at line 45 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 (
    uint8_t a_char,
    T_LCD_ram_area area )
```

Screen RAM write function.

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

##### Parameters

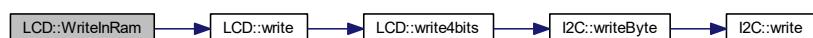
in	a_char	Data byte to write in RAM
in	area	Area in RAM where the data will be written : DDRAM or CGRAM

##### Returns

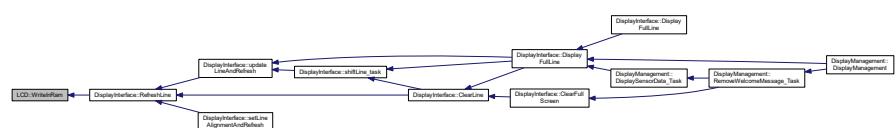
Nothing

Definition at line 187 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:

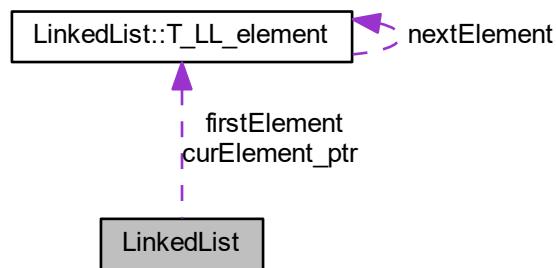
- [LCD.h](#)
- [LCD.cpp](#)

## 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

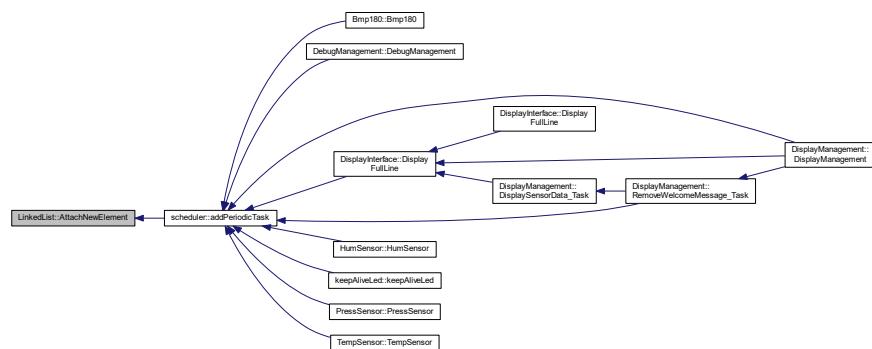
in	<i>data_ptr</i>	Pointer to the data element
----	-----------------	-----------------------------

#### 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**

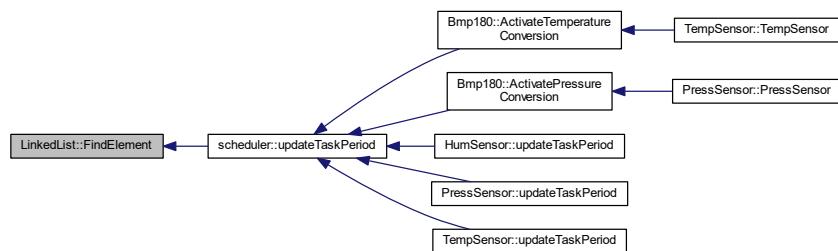
in	<i>comparisonFct_ptr</i>	Pointer to the comparison function
in	<i>reference_ptr</i>	Pointer to the element to find in the chain
out	<i>chainElement_ptr</i>	Pointer to pointer to the found element

**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.

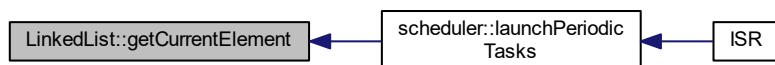
This function returns a pointer to the current pointed data in the chain.

**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.

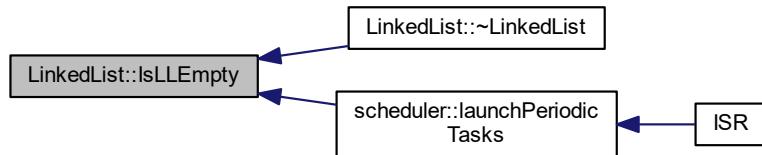
This function checks whether the linked list is empty or not (pointer to first element is equal to 0 or not).

##### 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.

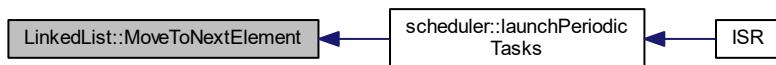
This function moves the element pointer to the next element of the chain.

##### 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

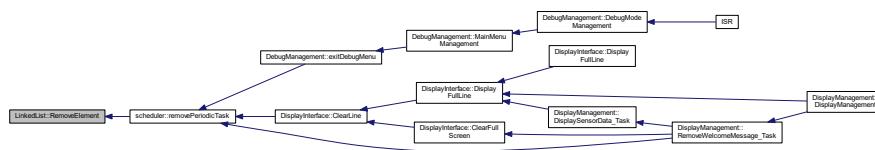
in	<i>comparisonFct_ptr</i>	Pointer to the comparison function to use
in	<i>reference_ptr</i>	Pointer to the reference data used for comparison

#### 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.

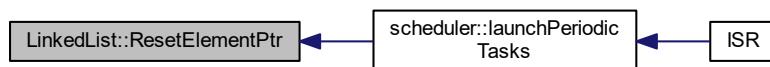
This function sets the element pointer to the first element of the chain.

#### 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:

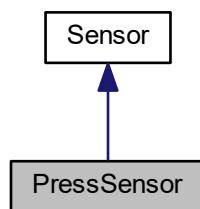
- [LinkedList.h](#)
- [LinkedList.cpp](#)

### 4.15 PressSensor Class Reference

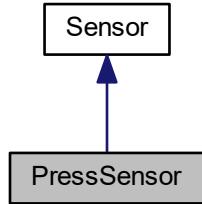
Class for pressure sensor.

```
#include <PressSensor.h>
```

Inheritance diagram for PressSensor:



Collaboration diagram for PressSensor:



## Public Member Functions

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

## Static Public Member Functions

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

## Additional Inherited Members

### 4.15.1 Detailed Description

Class for pressure sensor.

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

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

### 4.15.2 Constructor & Destructor Documentation

#### 4.15.2.1 PressSensor() [1/2]

```
PressSensor::PressSensor ( )
```

Class constructor.

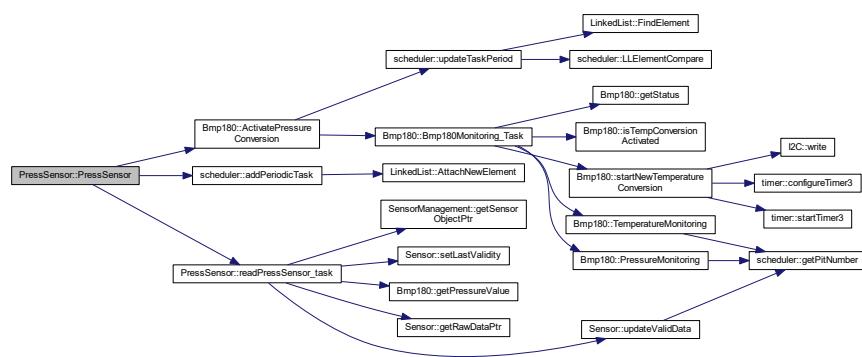
This function initializes all data of the class [PressSensor](#). If needed, it creates a new instance of the BMP180 sensor object. It also adds the periodic task in the scheduler.

#### Returns

Nothing

Definition at line 26 of file PressSensor.cpp.

Here is the call graph for this function:



#### 4.15.2.2 PressSensor() [2/2]

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

Overloaded class constructor.

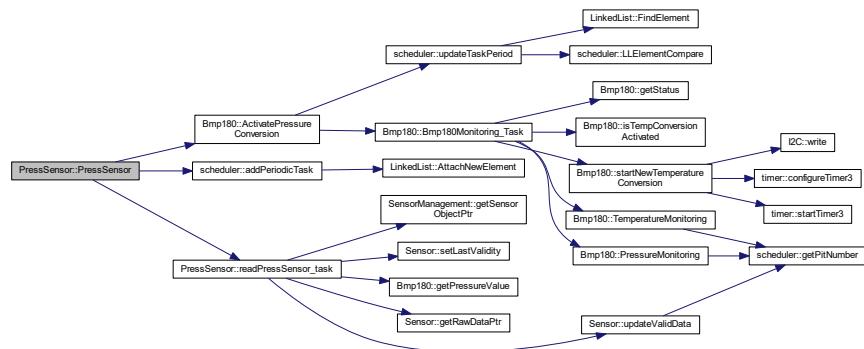
This function initializes all data of the class [PressSensor](#). It sets validity timeout and task period to the given value. If needed, it creates a new instance of the BMP180 sensor object. It also adds the periodic task in the scheduler.

## Returns

Nothing

Definition at line 38 of file PressSensor.cpp.

Here is the call graph for this function:



#### 4.15.3 Member Function Documentation

#### 4.15.3.1 `readPressSensor_task()`

```
void PressSensor::readPressSensor_task ( ) [static]
```

## Task for reading pressure values.

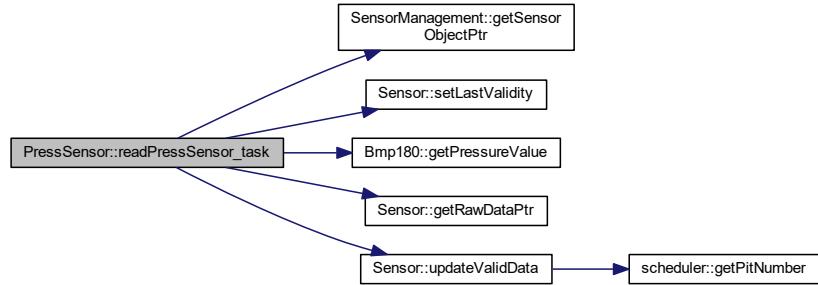
This task reads pressure data using BMP180 driver. It is called periodically.

## Returns

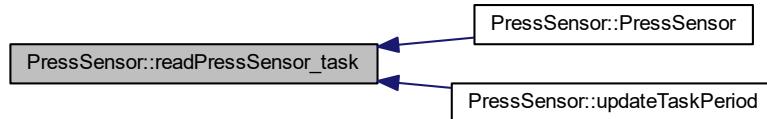
Nothing

Definition at line 50 of file PressSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.15.3.2 updateTaskPeriod()

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

Task period update.

This function updates the period of the pressure task.

##### Parameters

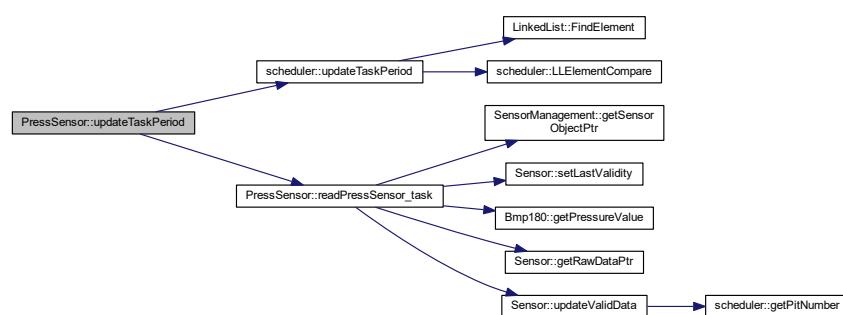
in	<code>period</code>	New period of the task
----	---------------------	------------------------

##### Returns

True if the period has been updated, false otherwise

Definition at line 64 of file `PressSensor.cpp`.

Here is the call graph for this function:



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

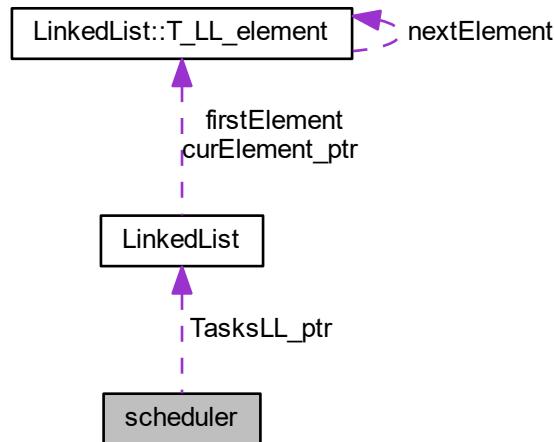
- [PressSensor.h](#)
- [PressSensor.cpp](#)

## 4.16 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](#)

### 4.16.1 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.

### 4.16.2 Member Typedef Documentation

#### 4.16.2.1 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.

### 4.16.3 Constructor & Destructor Documentation

#### 4.16.3.1 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:



#### 4.16.4 Member Function Documentation

##### 4.16.4.1 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

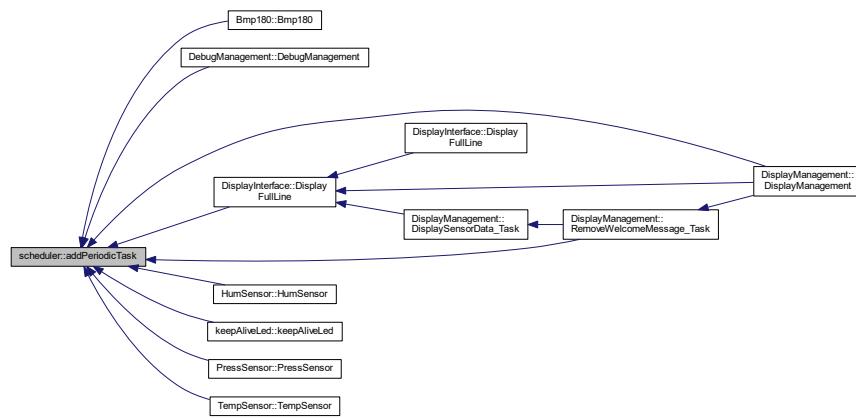
Nothing

Definition at line 99 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.16.4.2 getPitNumber()

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

Get function for PIT number.

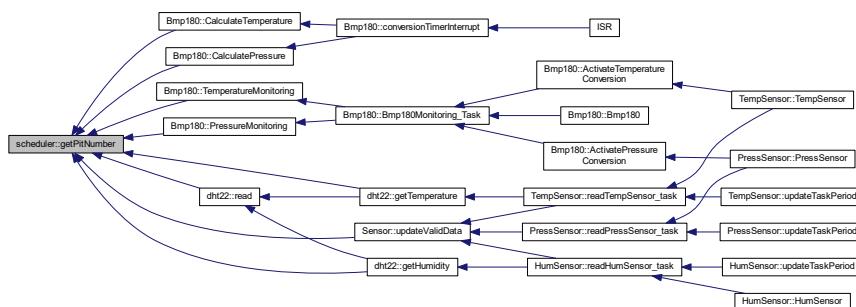
This function returns the PIT number

Returns

PIT number

Definition at line 113 of file scheduler.cpp.

Here is the caller graph for this function:



#### 4.16.4.3 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.

#### 4.16.4.4 launchPeriodicTasks()

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

Main scheduler function.

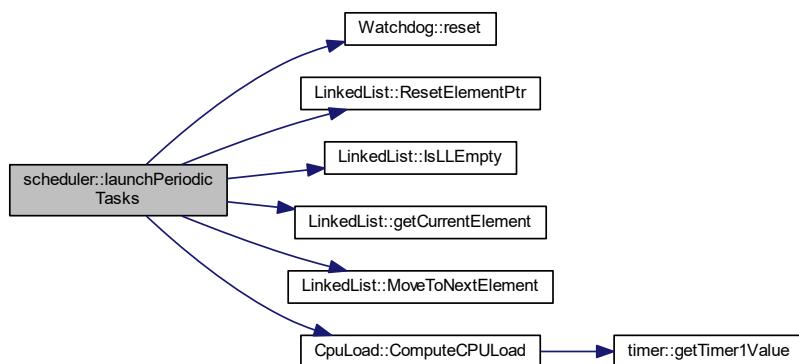
This function launches the scheduled tasks according to current software time and task configuration

##### Returns

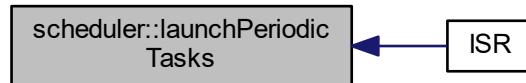
Nothing

Definition at line 54 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.16.4.5 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

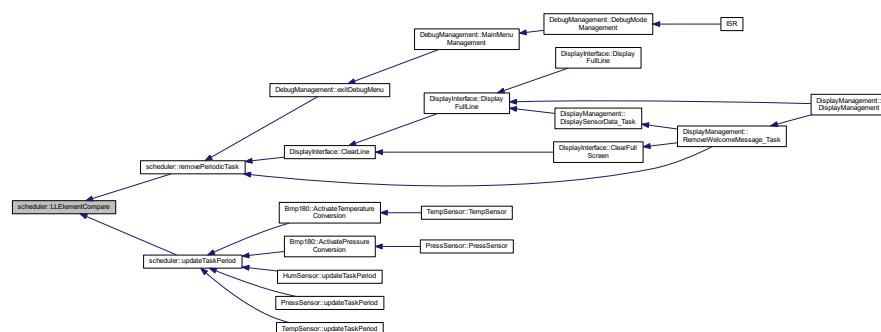
in	<i>LLElement</i>	Pointer to the linked list element
in	<i>CompareElement</i>	Pointer to the element to the compare

#### 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:



#### 4.16.4.6 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

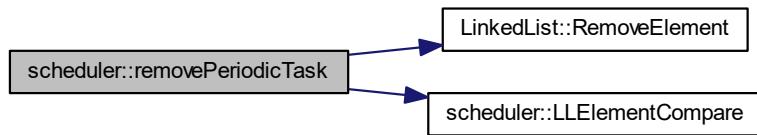
in	task_ptr	address of the task to remove from scheduler
----	----------	--

##### Returns

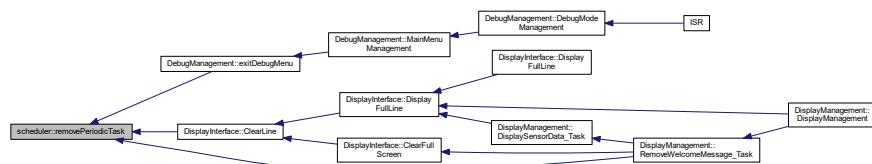
TRUE if the task has been removed, FALSE if the task does not exist in the scheduler

Definition at line 119 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.16.4.7 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**

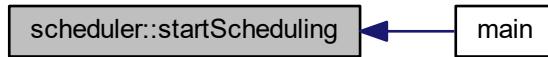
Nothing

Definition at line 93 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.16.4.8 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**

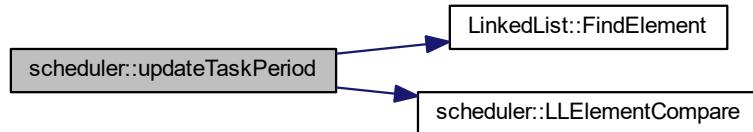
in	<i>task_ptr</i>	Pointer of the task to update
in	<i>period</i>	New period of the task

**Returns**

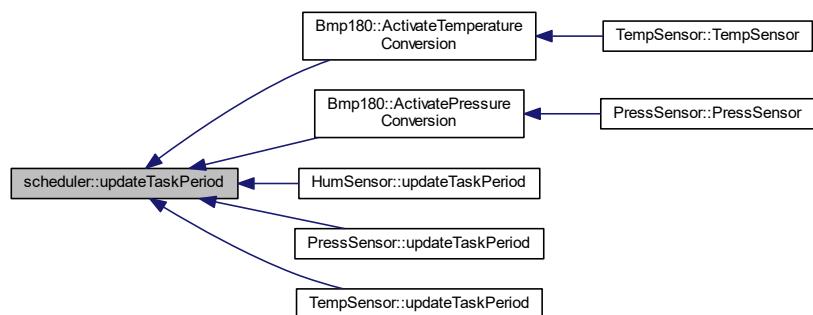
True if the update has been correctly done, false otherwise

Definition at line 142 of file scheduler.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.16.5 Member Data Documentation

### 4.16.5.1 pit\_number

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

Counter of periodic interrupts

Definition at line 140 of file scheduler.h.

### 4.16.5.2 task\_count

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

Number of task in scheduler

Definition at line 136 of file scheduler.h.

#### 4.16.5.3 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:

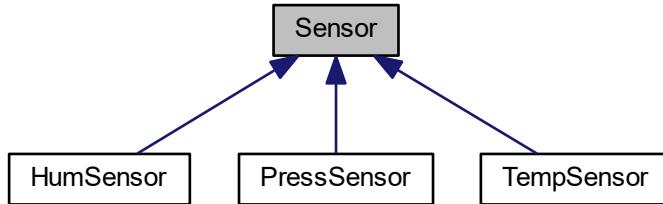
- [scheduler.h](#)
- [scheduler.cpp](#)

## 4.17 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`

### 4.17.1 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.

### 4.17.2 Constructor & Destructor Documentation

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

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

`Sensor` class constructor.

This function initializes the class.

**Returns**

Nothing

Definition at line 22 of file Sensor.cpp.

#### 4.17.2.2 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.

### 4.17.3 Member Function Documentation

#### 4.17.3.1 getRawDataPtr()

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

Get pointer to raw sensor data.

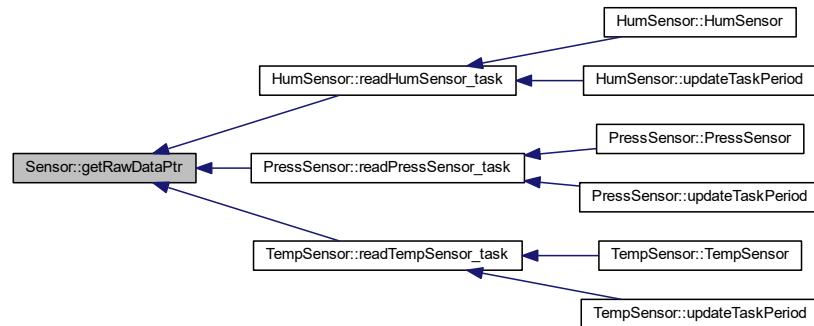
This function returns a pointer to the class member `raw_data`

##### Returns

Pointer to `raw_data`

Definition at line 53 of file Sensor.h.

Here is the caller graph for this function:



#### 4.17.3.2 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.

#### 4.17.3.3 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.

#### 4.17.3.4 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.

#### 4.17.3.5 `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.

#### 4.17.3.6 `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.

#### 4.17.3.7 `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.

#### 4.17.3.8 `setLastValidity()`

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

Validity setting function.

This function sets the class member `validity_last_read`

**Parameters**

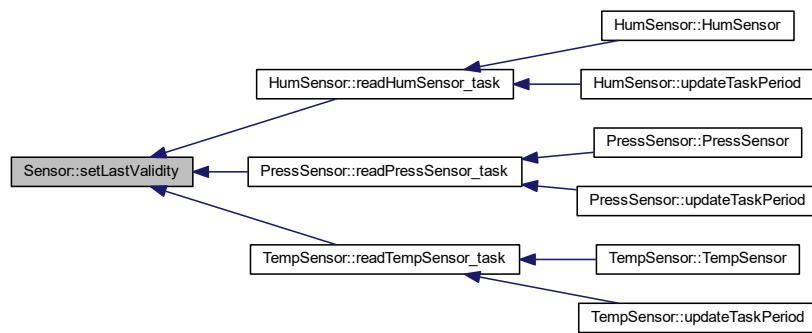
in	<i>validity</i>	Value of validity
----	-----------------	-------------------

**Returns**

Nothing

Definition at line 76 of file Sensor.h.

Here is the caller graph for this function:

**4.17.3.9 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.

#### 4.17.3.10 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.

#### 4.17.3.11 updateValidData()

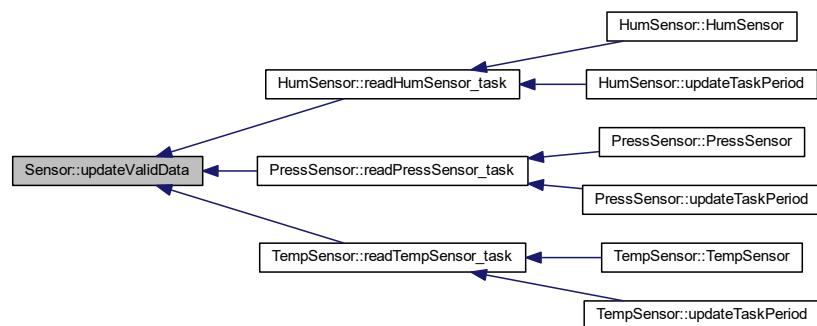
```
void Sensor::updateValidData ( )
```

Definition at line 53 of file Sensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.17.4 Member Data Documentation

### 4.17.4.1 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.

### 4.17.4.2 task\_period

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

Task period

Definition at line 165 of file Sensor.h.

### 4.17.4.3 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.

### 4.17.4.4 valid\_value

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

Valid value of sensor data

Definition at line 163 of file Sensor.h.

### 4.17.4.5 validity

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

Validity of sensor data

Definition at line 155 of file Sensor.h.

#### 4.17.4.6 validity\_last\_read

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

Validity of last read sensor data

Definition at line 156 of file Sensor.h.

#### 4.17.4.7 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](#)

## 4.18 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](#)

### 4.18.1 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 30 of file SensorManagement.h.

### 4.18.2 Constructor & Destructor Documentation

#### 4.18.2.1 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 27 of file SensorManagement.cpp.

### 4.18.3 Member Function Documentation

#### 4.18.3.1 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**

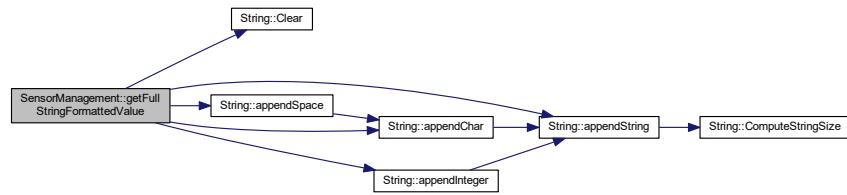
in	<i>sensor_idx</i>	Index of the requested sensor
out	<i>str</i>	Pointer to the formatted string

**Returns**

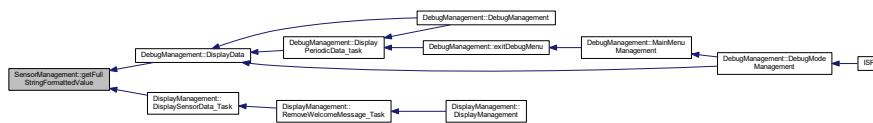
Nothing

Definition at line 77 of file SensorManagement.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.2 getSensorCount()

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

**Sensor** count get function.

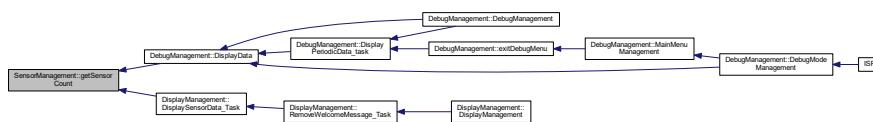
This function returns the number of sensors present in the SW

**Returns**

Number of sensors.

Definition at line 48 of file SensorManagement.h.

Here is the caller graph for this function:



#### 4.18.3.3 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**

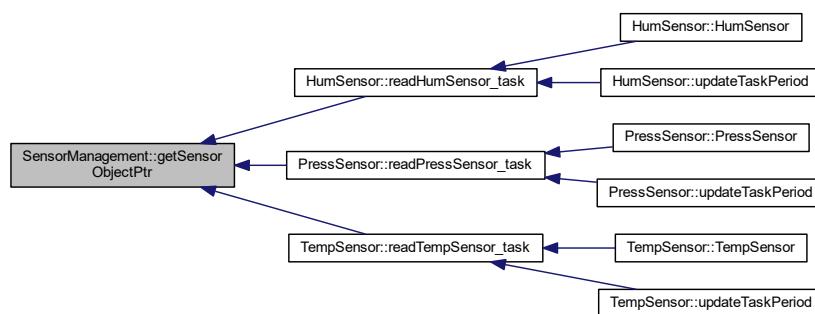
<code>in</code>	<code>type</code>	Type of sensor to find
-----------------	-------------------	------------------------

**Returns**

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

Definition at line 97 of file SensorManagement.cpp.

Here is the caller graph for this function:

**4.18.3.4 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 60 of file SensorManagement.cpp.

**4.18.4 Member Data Documentation**

#### 4.18.4.1 nb\_sensors

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

Number of sensors

Definition at line 83 of file SensorManagement.h.

#### 4.18.4.2 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 84 of file SensorManagement.h.

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

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

## 4.19 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`

### 4.19.1 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.

### 4.19.2 Constructor & Destructor Documentation

#### 4.19.2.1 `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:



#### 4.19.2.2 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.

#### 4.19.2.3 ~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:



### 4.19.3 Member Function Documentation

#### 4.19.3.1 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

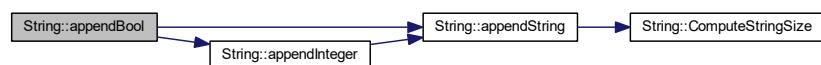
in	<code>data</code>	Boolean data to add
in	<code>isText</code>	Defines the conversion mode : text or integer

##### 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:



### 4.19.3.2 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

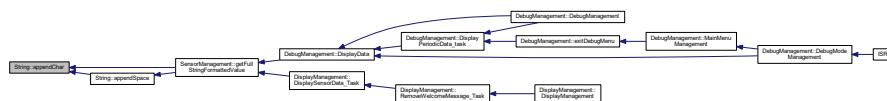
Nothing

Definition at line 139 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 4.19.3.3 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**

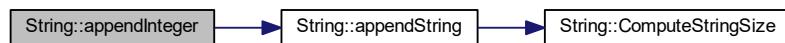
<i>in</i>	<i>value</i>	Integer to add
<i>in</i>	<i>base</i>	Base of computation of the integer (between 2 and 36)

**Returns**

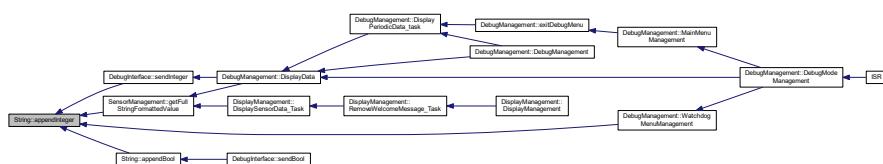
Nothing

Definition at line 95 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.19.3.4 appendSpace()**

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

Space adding function.

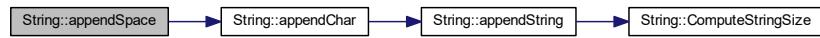
This function adds a space at the end of the string. It only calls appendChar function.

Returns

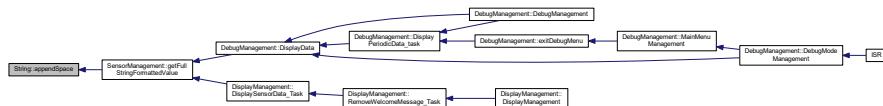
Nothing

Definition at line 123 of file String.h.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.19.3.5 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

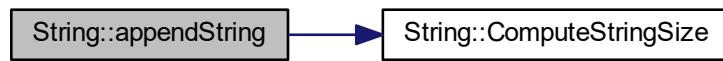
in	<i>str</i>	New string to add
----	------------	-------------------

**Returns**

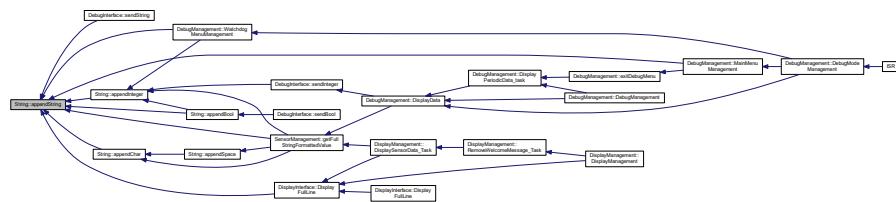
Nothing

Definition at line 57 of file String.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.19.3.6 Clear()**

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

`String` clear function.

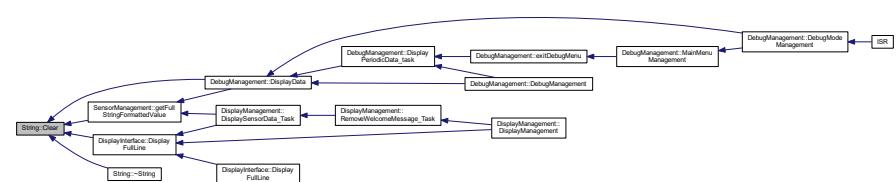
This function clears the string. Size is set to 0 and the memory is freed.

**Returns**

Nothing

Definition at line 113 of file String.cpp.

Here is the caller graph for this function:



#### 4.19.3.7 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

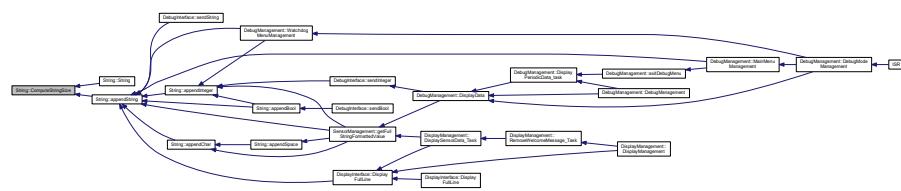
in	<i>str</i>	Pointer to the beginning of the string
----	------------	--

#### 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:



#### 4.19.3.8 getSize()

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

Size get function.

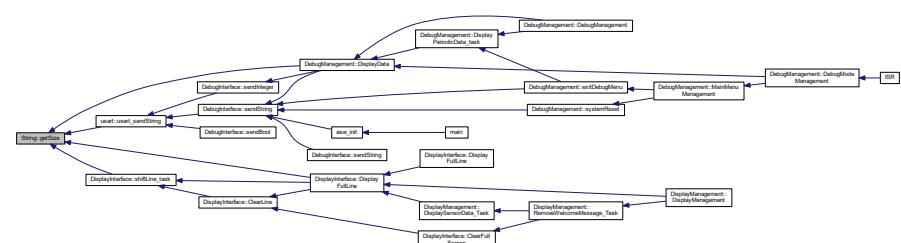
This function returns the size of the string.

#### Returns

Size of the string

Definition at line 64 of file String.h.

Here is the caller graph for this function:



#### 4.19.3.9 `getString()`

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

`String` pointer get function.

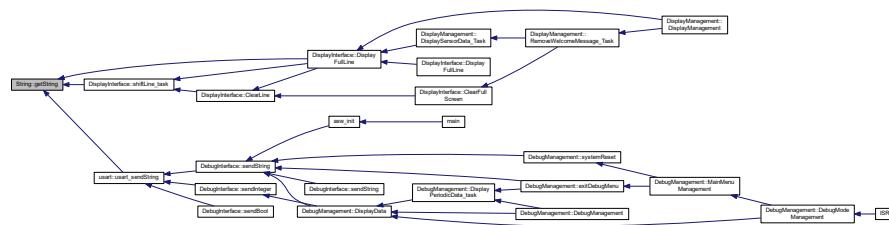
This function returns the pointer to the beginning of the string.

##### Returns

`String` pointer

Definition at line 53 of file `String.h`.

Here is the caller graph for this function:



#### 4.19.4 Member Data Documentation

##### 4.19.4.1 `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`.

##### 4.19.4.2 `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](#)

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

ASW initialization configuration structure.

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

### Public Attributes

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

#### 4.20.1 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.

#### 4.20.2 Member Data Documentation

##### 4.20.2.1 `isDebugActivated`

```
bool T_ASW_init_cnf::isDebugActivated
```

Debug services activation flag

Definition at line 19 of file asw.h.

##### 4.20.2.2 `isDisplayActivated`

```
bool T_ASW_init_cnf::isDisplayActivated
```

LCD display activation flag

Definition at line 22 of file asw.h.

#### 4.20.2.3 isLEDActivated

bool T\_ASW\_init\_cnf::isLEDActivated

Keep-alive LED activation flag

Definition at line 20 of file asw.h.

#### 4.20.2.4 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](#)

## 4.21 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

### 4.21.1 Detailed Description

Structure defining the calibration data of BMP180 sensor.

Definition at line 237 of file Bmp180.h.

## 4.21.2 Member Data Documentation

### 4.21.2.1 AC1

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

Definition at line 239 of file Bmp180.h.

### 4.21.2.2 AC2

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

Definition at line 240 of file Bmp180.h.

### 4.21.2.3 AC3

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

Definition at line 241 of file Bmp180.h.

### 4.21.2.4 AC4

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

Definition at line 242 of file Bmp180.h.

### 4.21.2.5 AC5

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

Definition at line 243 of file Bmp180.h.

#### 4.21.2.6 AC6

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

Definition at line 244 of file Bmp180.h.

#### 4.21.2.7 B1

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

Definition at line 245 of file Bmp180.h.

#### 4.21.2.8 B2

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

Definition at line 246 of file Bmp180.h.

#### 4.21.2.9 MB

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

Definition at line 247 of file Bmp180.h.

#### 4.21.2.10 MC

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

Definition at line 248 of file Bmp180.h.

#### 4.21.2.11 MD

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

Definition at line 249 of file Bmp180.h.

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

- [Bmp180.h](#)

## 4.22 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`

#### 4.22.1 Detailed Description

Structure defining a sensor value and its status.

Definition at line 258 of file Bmp180.h.

#### 4.22.2 Member Data Documentation

##### 4.22.2.1 ready

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

Measurement readiness flag

Definition at line 261 of file Bmp180.h.

##### 4.22.2.2 ts

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

Time stamp of last measurement

Definition at line 262 of file Bmp180.h.

##### 4.22.2.3 value

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

Last measurement value

Definition at line 260 of file Bmp180.h.

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

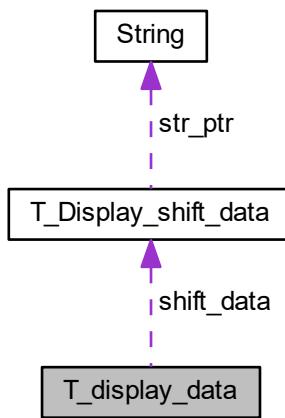
- [Bmp180.h](#)

## 4.23 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]`

#### 4.23.1 Detailed Description

Structure containing display data.

This structure contains all data used for screen display

Definition at line 57 of file `DisplayInterface.h`.

#### 4.23.2 Member Data Documentation

#### 4.23.2.1 alignment

`T_DisplayInterface_LineAlignment` `T_display_data::alignment`

Line alignment

Definition at line 61 of file DisplayInterface.h.

#### 4.23.2.2 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.

#### 4.23.2.3 isEmpty

`bool` `T_display_data::isEmpty`

Flag indicating if the line is empty or not

Definition at line 59 of file DisplayInterface.h.

#### 4.23.2.4 mode

`T_DisplayInterface_LineDisplayMode` `T_display_data::mode`

Current display mode

Definition at line 60 of file DisplayInterface.h.

#### 4.23.2.5 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:

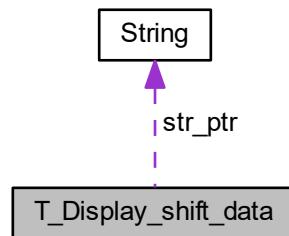
- [DisplayInterface.h](#)

## 4.24 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`

#### 4.24.1 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.

#### 4.24.2 Member Data Documentation

##### 4.24.2.1 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.

#### 4.24.2.2 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.

#### 4.24.2.3 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](#)

## 4.25 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](#)

### 4.25.1 Detailed Description

Structure defining [LCD](#) configuration.

Definition at line 128 of file LCD.h.

## 4.25.2 Member Data Documentation

### 4.25.2.1 `backlight_en`

```
bool T_LCD_conf_struct::backlight_en
```

Screen backlight enable flag

Definition at line 132 of file LCD.h.

### 4.25.2.2 `cursor_en`

```
bool T_LCD_conf_struct::cursor_en
```

Screen cursor enable flag

Definition at line 136 of file LCD.h.

### 4.25.2.3 `cursorBlink_en`

```
bool T_LCD_conf_struct::cursorBlink_en
```

Screen cursor blinking enable flag

Definition at line 137 of file LCD.h.

### 4.25.2.4 `display_en`

```
bool T_LCD_conf_struct::display_en
```

Screen display enable flag

Definition at line 135 of file LCD.h.

### 4.25.2.5 `entryModeDir`

```
bool T_LCD_conf_struct::entryModeDir
```

Entry mode direction configuration

Definition at line 138 of file LCD.h.

#### 4.25.2.6 entryModeShift

```
bool T_LCD_conf_struct::entryModeShift
```

Entry mode shift configuration

Definition at line 139 of file LCD.h.

#### 4.25.2.7 fontType\_cnf

```
bool T_LCD_conf_struct::fontType_cnf
```

Font configuration

Definition at line 134 of file LCD.h.

#### 4.25.2.8 i2c\_addr

```
uint8_t T_LCD_conf_struct::i2c_addr
```

I<sup>2</sup>C address if the screen

Definition at line 131 of file LCD.h.

#### 4.25.2.9 i2c\_bitrate

```
uint32_t T_LCD_conf_struct::i2c_bitrate
```

I<sup>2</sup>C bitrate needed by the LCD screen

Definition at line 130 of file LCD.h.

#### 4.25.2.10 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.

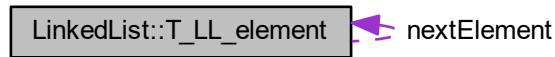
The documentation for this struct was generated from the following file:

- [LCD.h](#)

## 4.26 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](#)

#### 4.26.1 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](#).

#### 4.26.2 Member Data Documentation

##### 4.26.2.1 [data\\_ptr](#)

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

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

##### 4.26.2.2 [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](#)

## 4.27 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`

### 4.27.1 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`.

### 4.27.2 Member Data Documentation

#### 4.27.2.1 data\_name\_str

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

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

#### 4.27.2.2 period

```
uint16_t T_SensorManagement_Sensor_Config::period
```

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

#### 4.27.2.3 sensor\_type

```
T_SensorManagement_Sensor_Type T_SensorManagement_Sensor_Config::sensor_type
```

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

#### 4.27.2.4 unit\_str

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

Definition at line 23 of file sensor\_configuration.h.

#### 4.27.2.5 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](#)

## 4.28 scheduler::Task\_t Struct Reference

Type defining a task structure.

### Public Attributes

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

#### 4.28.1 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.

#### 4.28.2 Member Data Documentation

##### 4.28.2.1 period

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

Period of the task

Definition at line 132 of file scheduler.h.

### 4.28.2.2 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:

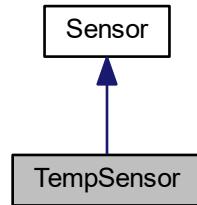
- [scheduler.h](#)

## 4.29 TempSensor Class Reference

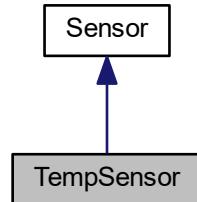
Class for temperature sensor.

```
#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

### 4.29.1 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.

### 4.29.2 Constructor & Destructor Documentation

#### 4.29.2.1 [TempSensor\(\)](#) [1/2]

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

Class constructor.

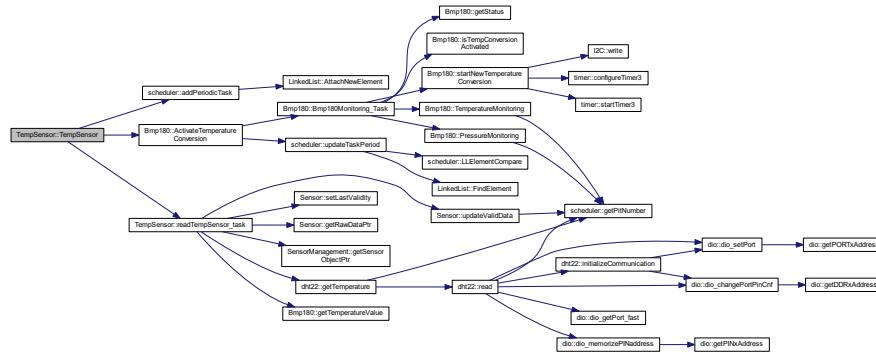
This function initializes all data of the class [TempSensor](#). If needed, it creates a new instance of the DHT22 and BMP180 sensors objects. It also adds the periodic task in the scheduler.

**Returns**

Nothing

Definition at line 29 of file TempSensor.cpp.

Here is the call graph for this function:

**4.29.2.2 TempSensor() [2/2]**

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

Overloaded class constructor.

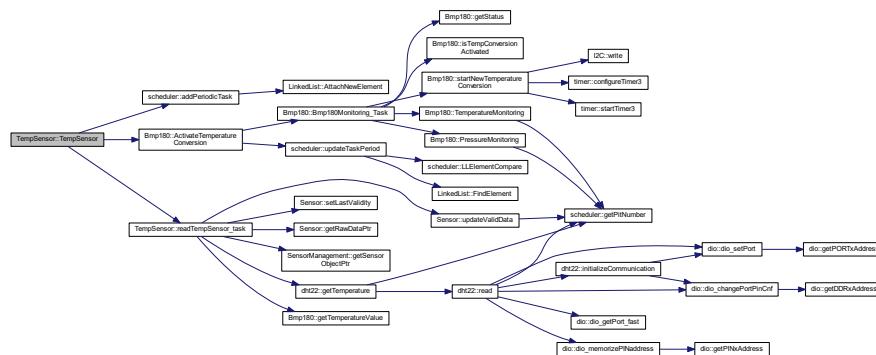
This function initializes all data of the class [TempSensor](#). It sets validity timeout and task period to the given value. If needed, it creates a new instance of the DHT22 and BMP180 sensor objects. It also adds the periodic task in the scheduler.

**Returns**

Nothing

Definition at line 46 of file TempSensor.cpp.

Here is the call graph for this function:



### 4.29.3 Member Function Documentation

#### 4.29.3.1 readTempSensor\_task()

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

## Task for reading temperature values.

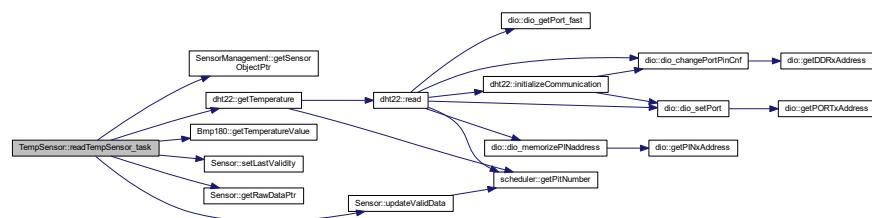
This task reads temperature data using DHT22 and BMP180 drivers. It is called periodically. The returned temperature is the mean between both sensors values, if only one sensor is valid, only this value is used .

## Returns

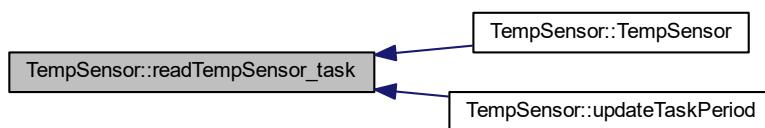
Nothing

Definition at line 62 of file TempSensor.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.29.3.2 updateTaskPeriod()

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

## Task period update.

This function updates the period of the temperature task.

## Parameters

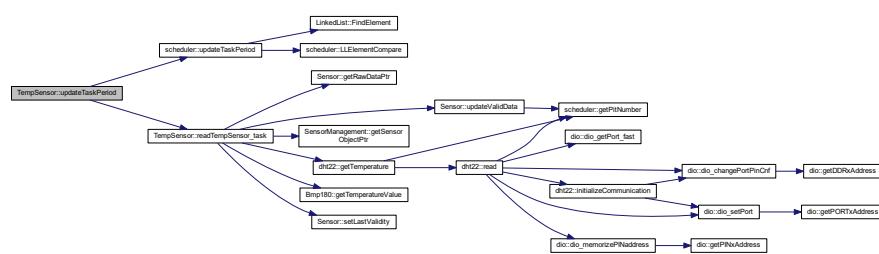
in	<i>period</i>	New period of the task
----	---------------	------------------------

## Returns

True if the period has been updated, false otherwise

Definition at line 114 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](#)

## 4.30 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`

### 4.30.1 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.

### 4.30.2 Constructor & Destructor Documentation

#### 4.30.2.1 timer()

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

Class constructor.

This function initializes class attributes

Returns

Nothing

Definition at line 15 of file timer.cpp.

### 4.30.3 Member Function Documentation

#### 4.30.3.1 configureTimer1()

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

Configures Timer #1.

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

**Parameters**

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

**Returns**

Nothing

Definition at line 21 of file timer.cpp.

Here is the caller graph for this function:

**4.30.3.2 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**

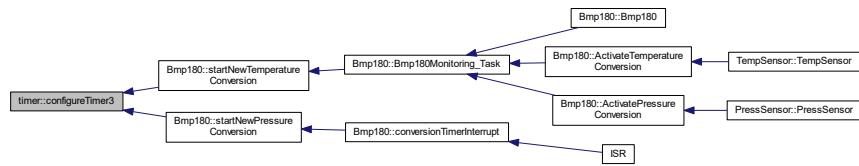
in	<i>a_prescaler</i>	prescaler value
in	<i>a_ctcValue</i>	Value to which the counter will compare before raising an interrupt

**Returns**

Nothing

Definition at line 58 of file timer.cpp.

Here is the caller graph for this function:



#### 4.30.3.3 `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:



#### 4.30.3.4 `startTimer1()`

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

Start Timer #1.

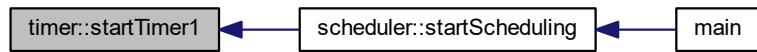
This functions starts Timer #1. Timer shall be initialized before this function is called.

**Returns**

Nothing

Definition at line 96 of file timer.cpp.

Here is the caller graph for this function:

**4.30.3.5 startTimer3()**

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

Start Timer #3.

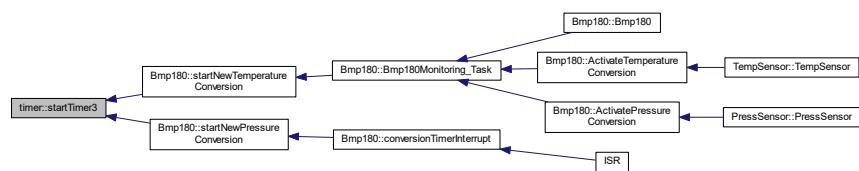
This functions starts Timer #3. Timer shall be initialized before this function is called.

**Returns**

Nothing

Definition at line 107 of file timer.cpp.

Here is the caller graph for this function:

**4.30.3.6 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.

#### 4.30.3.7 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:



#### 4.30.4 Member Data Documentation

##### 4.30.4.1 prescaler1

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

Definition at line 87 of file timer.h.

##### 4.30.4.2 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](#)

## 4.31 usart Class Reference

USART serial bus class.

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

### Public Member Functions

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

### Private Member Functions

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

### Private Attributes

- uint16\_t **BaudRate**

#### 4.31.1 Detailed Description

USART serial bus class.

This class defines all useful functions for USART serial bus

Definition at line 16 of file usart.h.

#### 4.31.2 Constructor & Destructor Documentation

##### 4.31.2.1 usart()

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

Class usart constructor.

Initializes the class and call hardware initialization function

**Parameters**

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

**Returns**

Nothing.

Definition at line 18 of file usart.cpp.

Here is the call graph for this function:



### 4.31.3 Member Function Documentation

#### 4.31.3.1 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.

#### 4.31.3.2 usart\_init()

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

USART hardware initialization.

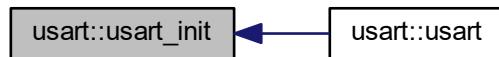
This function will initialize the USART using selected baudrate. User must pay attention to select one of the usually used Baud Rate (9600, 19200, 38400, 57600). Note that since an uint16 is used as argument, Baud rate cannot be more than 57600.

##### Returns

Nothing.

Definition at line 25 of file usart.cpp.

Here is the caller graph for this function:



#### 4.31.3.3 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.

#### 4.31.3.4 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 *data* 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:



#### 4.31.3.5 usart\_sendString()

```
void usart::usart_sendString (
```

Send a string on USART link.

This function writes the string object data to the serial link using usart\_transmit function

## Parameters

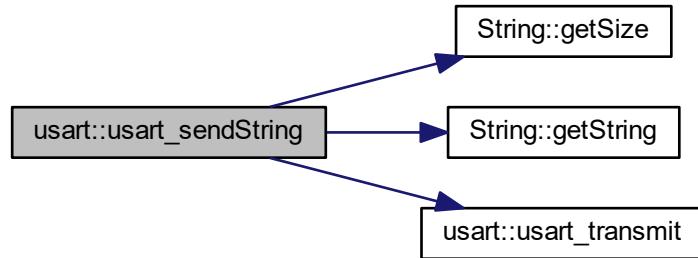
in *str* Pointer to the string being sent

## Returns

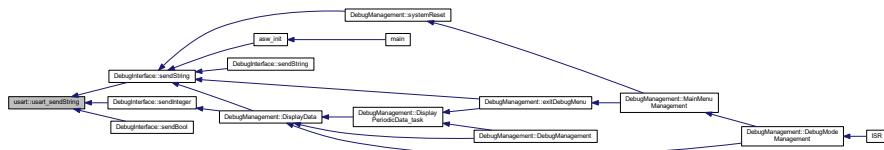
Nothing.

Definition at line 48 of file usart.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.31.3.6 usart\_transmit()

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

USART Transmit data.

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

##### Parameters

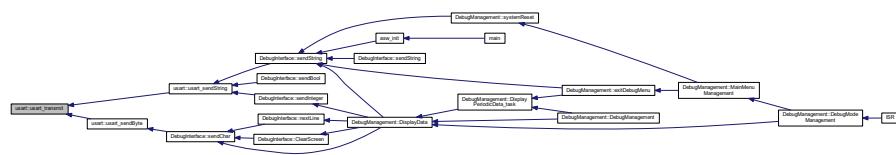
in	<i>Data</i>	Desired data char to transmit
----	-------------	-------------------------------

##### Returns

Nothing.

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

Here is the caller graph for this function:



#### 4.31.4 Member Data Documentation

##### 4.31.4.1 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:

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

## 4.32 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**

### 4.32.1 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.

### 4.32.2 Constructor & Destructor Documentation

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

`Watchdog::Watchdog ( )`

Class constructor.

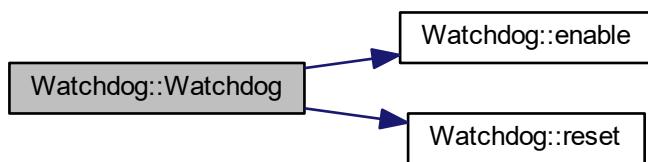
This function initializes the watchdog class. It enables the HW watchdog with a default timeout value.

#### Returns

Nothing

Definition at line 19 of file Watchdog.cpp.

Here is the call graph for this function:



### 4.32.2.2 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

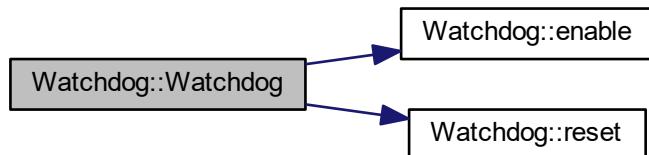
<code>in</code>	<code>timeout</code>	Timeout value requested for the watchdog
-----------------	----------------------	--

#### Returns

Nothing

Definition at line 26 of file Watchdog.cpp.

Here is the call graph for this function:



### 4.32.3 Member Function Documentation

#### 4.32.3.1 disable()

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

[Watchdog](#) disabling function.

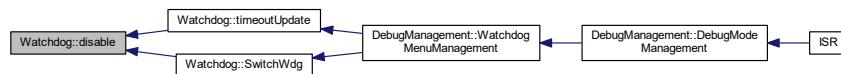
This function disables the watchdog by calling `wdt_disable` macro.

**Returns**

Nothing

Definition at line 44 of file Watchdog.cpp.

Here is the caller graph for this function:

**4.32.3.2 enable()**

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

[Watchdog](#) enabling function.

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

**Parameters**

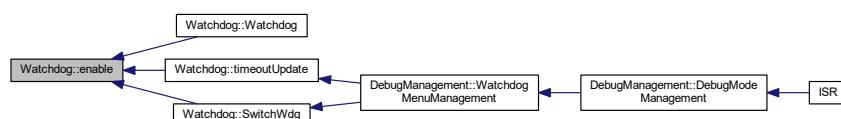
in	value	Timeout value
----	-------	---------------

**Returns**

Nothing

Definition at line 34 of file Watchdog.cpp.

Here is the caller graph for this function:



#### 4.32.3.3 getTMOValue()

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

**Watchdog** timeout get value.

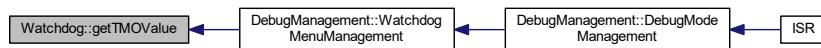
This function returns the current watchdog timeout value in ms. It has to convert the value of tmo\_value into a numeric value of the timeout.

##### Returns

Timeout value.

Definition at line 75 of file Watchdog.cpp.

Here is the caller graph for this function:



#### 4.32.3.4 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:



#### 4.32.3.5 reset()

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

[Watchdog](#) reset function.

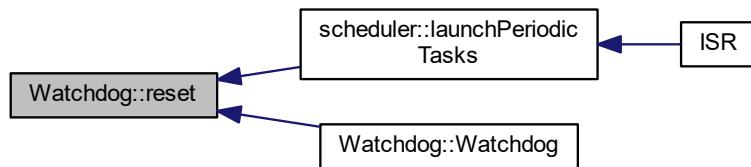
This function resets the watchdog timer by calling wdt\_reset macro

##### Returns

Nothing

Definition at line 53 of file Watchdog.cpp.

Here is the caller graph for this function:



#### 4.32.3.6 SwitchWdg()

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

[Watchdog](#) switching function.

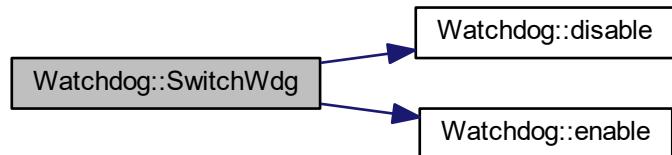
This function switches the state of the watchdog. If it was enabled, the function disables the watchdog, and if it was disabled, the function enables it with the memorized timeout value. The function returns the new status of the watchdog.

**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:

**4.32.3.7 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:



### 4.32.3.8 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

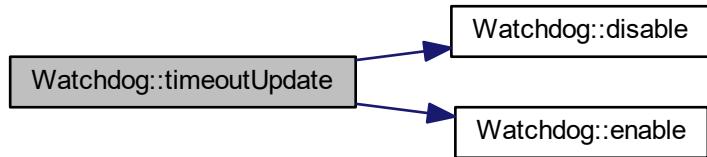
in	value	New timeout value
----	-------	-------------------

#### Returns

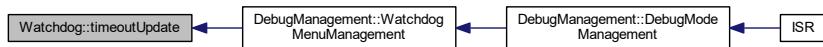
Nothing

Definition at line 58 of file Watchdog.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.32.4 Member Data Documentation

### 4.32.4.1 isActive

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

[Watchdog](#) activation flag

Definition at line 109 of file Watchdog.h.

#### 4.32.4.2 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

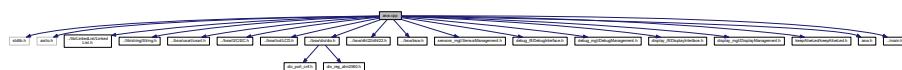
# File Documentation

### 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:



### 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.

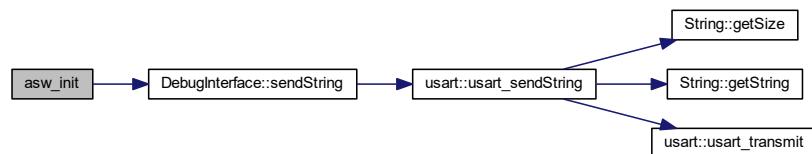
This function shall be called after BSW initialization function.

#### Returns

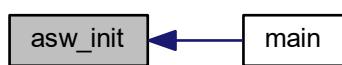
Nothing

Definition at line 35 of file asw.cpp.

Here is the call graph for this function:



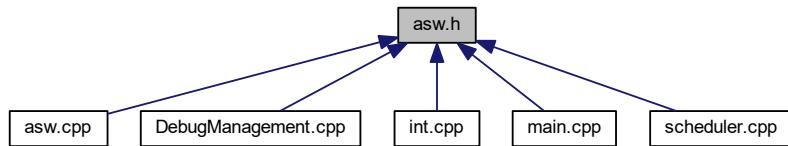
Here is the caller graph for this function:



## 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.

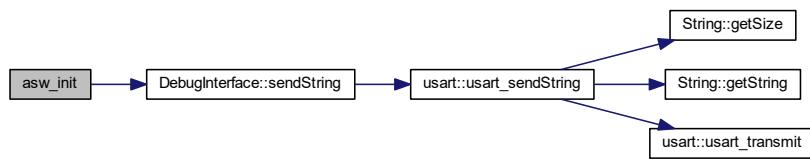
This function shall be called after BSW initialization function.

#### 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:

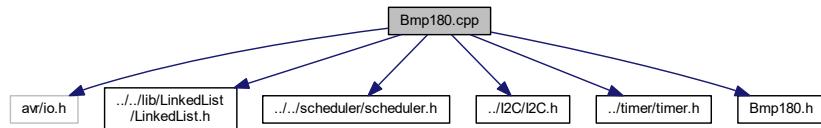


## 5.3 Bmp180.cpp File Reference

[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`

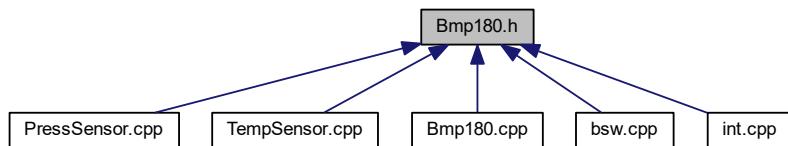
Pointer to BMP180 driver object

Definition at line 19 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\\_CTRL\\_MEAS\\_START\\_PRESS\\_CONV\\_OSS0](#) 0x34
- #define [BMP180\\_TEMP\\_MEAS\\_WAITING\\_TIME](#) 6
- #define [BMP180\\_PRESS\\_MEAS\\_OSS0\\_WAITING\\_TIME](#) 15
- #define [BMP180\\_TIMER\\_PRESCALER\\_VALUE](#) 64
- #define [BMP180\\_TEMP\\_MEAS\\_TIMER\\_CTC\\_VALUE](#) ((F\_CPU/BMP180\_TIMER\_PRESCALER\_VALUE)/(1000/BMP180\_TEMP\_MEAS\_WAITING\_TIME))
- #define [BMP180\\_PRESS\\_MEAS\\_OSS0\\_TIMER\\_CTC\\_VALUE](#) ((F\_CPU/BMP180\_TIMER\_PRESCALER\_VALUE)/(1000/BMP180\_PRESS\_MEAS\_WAITING\_TIME))
- #define [BMP180\\_OUT\\_REG\\_LSB EEPROM\\_ADDR](#) 0xF7
- #define [BMP180\\_OUT\\_REG\\_MSB EEPROM\\_ADDR](#) 0xF6
- #define [BMP180\\_MONITORING\\_DEFAULT\\_PERIOD](#) 500

## Enumerations

- enum [T\\_BMP180\\_status](#) { IDLE, TEMP\_CONV\_IN\_PROGRESS, PRESSURE\_CONV\_IN\_PROGRESS, COMM\_FAILED }
- Enumeration defining the possible statuses for BMP180 sensor driver.*

## Variables

- [Bmp180 \\* p\\_global\\_BSW\\_bmp180](#)

### 5.4.1 Detailed Description

[Bmp180](#) class header file.

#### Date

27 juil. 2019

#### Author

nicls67

## 5.4.2 Macro Definition Documentation

### 5.4.2.1 BMP180\_CHIP\_ID\_CALIB\_EEP\_START\_ADDR

```
#define BMP180_CHIP_ID_CALIB_EEP_START_ADDR 0xAA
```

EEPROM calibration data start address

Definition at line 17 of file Bmp180.h.

### 5.4.2.2 BMP180\_CHIP\_ID\_EEP\_ADDR

```
#define BMP180_CHIP_ID_EEP_ADDR 0xD0
```

Chip ID EEPROM address

Definition at line 16 of file Bmp180.h.

### 5.4.2.3 BMP180\_CHIP\_ID\_EXPECTED

```
#define BMP180_CHIP_ID_EXPECTED 0x55
```

Expected chip ID

Definition at line 18 of file Bmp180.h.

### 5.4.2.4 BMP180\_CTRL\_MEAS\_EEP\_ADDR

```
#define BMP180_CTRL_MEAS_EEP_ADDR 0xF4
```

Address of the measurement control register in EEPROM

Definition at line 20 of file Bmp180.h.

### 5.4.2.5 BMP180\_CTRL\_MEAS\_START\_PRESS\_CONV\_OSS0

```
#define BMP180_CTRL_MEAS_START_PRESS_CONV_OSS0 0x34
```

Value of measurement control register to start a pressure conversion with OSS0 parameter

Definition at line 22 of file Bmp180.h.

#### 5.4.2.6 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.

#### 5.4.2.7 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\_MONITORING\_DEFAULT\_PERIOD

```
#define BMP180_MONITORING_DEFAULT_PERIOD 500
```

Monitoring period is set by default to 500ms

Definition at line 34 of file Bmp180.h.

#### 5.4.2.10 BMP180\_OUT\_REG\_LSB\_EEPROM\_ADDR

```
#define BMP180_OUT_REG_LSB_EEPROM_ADDR 0xF7
```

Address of LSB out register

Definition at line 31 of file Bmp180.h.

**5.4.2.11 BMP180\_OUT\_REG\_MSB\_EEPROM\_ADDR**

```
#define BMP180_OUT_REG_MSB_EEPROM_ADDR 0xF6
```

Address of MSB out register

Definition at line 32 of file Bmp180.h.

**5.4.2.12 BMP180\_PRESS\_MEAS\_OSS0\_TIMER\_CTC\_VALUE**

```
#define BMP180_PRESS_MEAS_OSS0_TIMER_CTC_VALUE ((F_CPU/BMP180_TIMER_PRESCALER_VALUE) / (1000/BMP180_PRESS_MEAS_OSS0_WAITING_TIME))
```

Compare value for periodic timer in case of pressure conversion with parameter OSS0

Definition at line 29 of file Bmp180.h.

**5.4.2.13 BMP180\_PRESS\_MEAS\_OSS0\_WAITING\_TIME**

```
#define BMP180_PRESS_MEAS_OSS0_WAITING_TIME 15
```

Waiting time for a pressure conversion with parameter OSS0

Definition at line 25 of file Bmp180.h.

**5.4.2.14 BMP180\_TEMP\_MEAS\_TIMER\_CTC\_VALUE**

```
#define BMP180_TEMP_MEAS_TIMER_CTC_VALUE ((F_CPU/BMP180_TIMER_PRESCALER_VALUE) / (1000/BMP180_TEMP_MEAS_WAITING_TIME))
```

Compare value for periodic timer in case of temperature conversion

Definition at line 28 of file Bmp180.h.

**5.4.2.15 BMP180\_TEMP\_MEAS\_WAITING\_TIME**

```
#define BMP180_TEMP_MEAS_WAITING_TIME 6
```

Waiting time for a temperature conversion

Definition at line 24 of file Bmp180.h.

#### 5.4.2.16 BMP180\_TIMER\_PRESCALER\_VALUE

```
#define BMP180_TIMER_PRESCALER_VALUE 64
```

Value of prescaler to use for timer

Definition at line 27 of file Bmp180.h.

### 5.4.3 Enumeration Type Documentation

#### 5.4.3.1 T\_BMP180\_status

```
enum T_BMP180_status
```

Enumeration defining the possible statuses for BMP180 sensor driver.

**Enumerator**

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

Definition at line 39 of file Bmp180.h.

### 5.4.4 Variable Documentation

#### 5.4.4.1 p\_global\_BSW\_bmp180

```
Bmp180* p_global_BSW_bmp180
```

Pointer to BMP180 driver object

Definition at line 19 of file Bmp180.cpp.

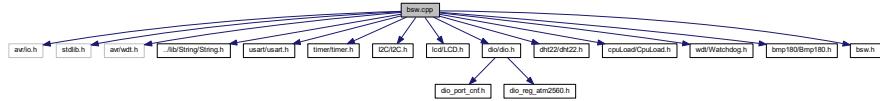
## 5.5 bsw.cpp File Reference

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.

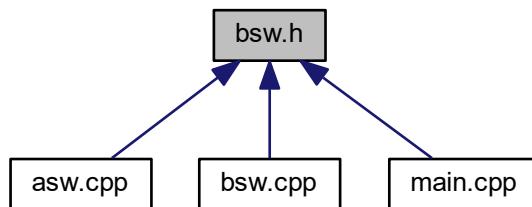
Here is the caller graph for this function:



## 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.

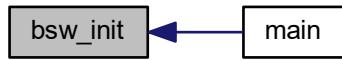
This function instantiates all driver objects, leading hardware initialization. The addresses of driver objects are then stored in BSW\_cnf\_struct structure.

#### Returns

Nothing

Definition at line 27 of file bsw.cpp.

Here is the caller graph for this function:

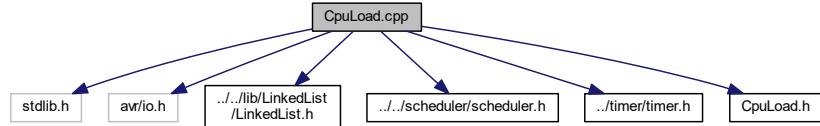


## 5.7 CpuLoad.cpp File Reference

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`

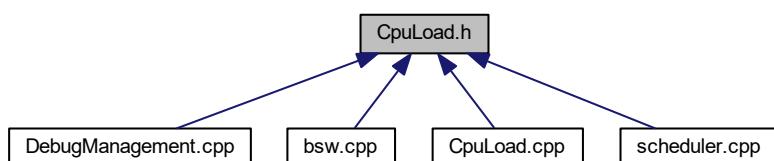
Pointer to cpu load library object

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

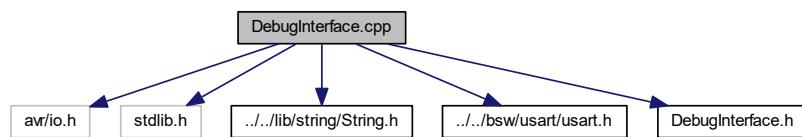
Definition at line 18 of file CpuLoad.cpp.

## 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`

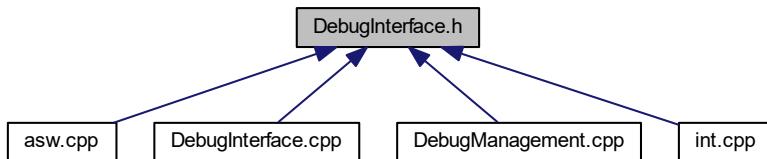
Pointer to USART debug interface object

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:



## Variables

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

### 5.11.1 Detailed Description

Debug management class source file.

#### Date

8 mai 2019

#### Author

nicls67

### 5.11.2 Variable Documentation

#### 5.11.2.1 p\_global\_ASW\_DebugManagement

`DebugManagement * p_global_ASW_DebugManagement`

Pointer to the `DebugManagement` object

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

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

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

Info menu string displayed when the watchdog has been disabled.

Definition at line 94 of file DebugManagement.cpp.

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

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

Info menu string displayed when the watchdog has been enabled.

Definition at line 99 of file DebugManagement.cpp.

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

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

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

Definition at line 84 of file DebugManagement.cpp.

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

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

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

Definition at line 89 of file DebugManagement.cpp.

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

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

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

Definition at line 79 of file DebugManagement.cpp.

### 5.11.2.7 str\_debug\_main\_menu

```
const uint8_t str_debug_main_menu[ ]
```

#### Initial value:

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

Main menu of debug mode.

Definition at line 40 of file DebugManagement.cpp.

### 5.11.2.8 str\_debug\_wdg\_menu

```
const uint8_t str_debug_wdg_menu[ ]
```

#### Initial value:

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

Watchdog menu of debug mode.

Definition at line 50 of file DebugManagement.cpp.

### 5.11.2.9 str\_debug\_wdg\_timeout\_update\_selection

```
const uint8_t str_debug_wdg_timeout_update_selection[ ]
```

#### Initial value:

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

Watchdog timeout update selection.

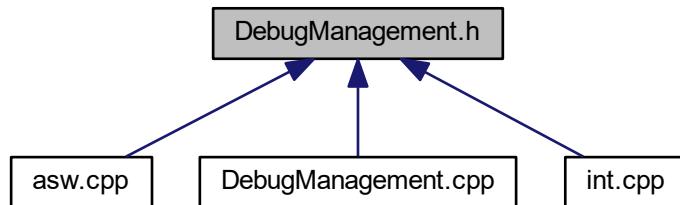
Definition at line 61 of file DebugManagement.cpp.

---

## 5.12 DebugManagement.h File Reference

Debug management class header file.

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



### Classes

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

### Macros

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

### Enumerations

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

### Variables

- `DebugManagement * p_global_ASW_DebugManagement`

#### 5.12.1 Detailed Description

Debug management class header file.

##### Date

8 mai 2019

##### Author

nicls67

## 5.12.2 Macro Definition Documentation

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

```
#define PERIOD_MS_TASK_DISPLAY_CPU_LOAD 5000
```

Period for displaying CPU load data

Definition at line 14 of file DebugManagement.h.

### 5.12.2.2 PERIOD\_MS\_TASK\_DISPLAY\_DEBUG\_DATA

```
#define PERIOD_MS_TASK_DISPLAY_DEBUG_DATA 5000
```

Period for displaying temperature and humidity data

Definition at line 13 of file DebugManagement.h.

## 5.12.3 Enumeration Type Documentation

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

```
enum debug_mgt_main_menu_state_t
```

Defines the debug states.

Enumerator

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

Definition at line 20 of file DebugManagement.h.

### 5.12.3.2 debug\_mgt\_wdg\_state\_t

```
enum debug_mgt_wdg_state_t
```

Defines possible states for watchdog management.

**Enumerator**

WDG_MAIN	Main menu of watchdog management
WDG_TMO_UPDATE	Timeout update mode

Definition at line 30 of file DebugManagement.h.

#### 5.12.4 Variable Documentation

##### 5.12.4.1 p\_global\_ASW\_DebugManagement

`DebugManagement* p_global_ASW_DebugManagement`

Pointer to the `DebugManagement` object

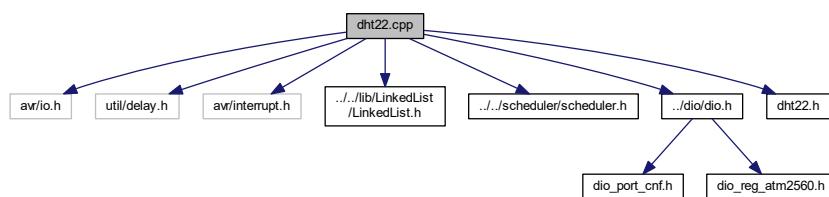
Definition at line 33 of file DebugManagement.cpp.

### 5.13 dht22.cpp File Reference

This file defines classes for DHT22 driver.

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

Include dependency graph for dht22.cpp:



### Macros

- `#define MAX_WAIT_TIME_US 100`

## Variables

- `dht22 * p_global_BSW_dht22`

### 5.13.1 Detailed Description

This file defines classes for DHT22 driver.

#### Date

23 mars 2018

#### Author

nicls67

### 5.13.2 Macro Definition Documentation

#### 5.13.2.1 MAX\_WAIT\_TIME\_US

```
#define MAX_WAIT_TIME_US 100
```

Maximum waiting time in microseconds

Definition at line 20 of file dht22.cpp.

### 5.13.3 Variable Documentation

#### 5.13.3.1 p\_global\_BSW\_dht22

```
dht22* p_global_BSW_dht22
```

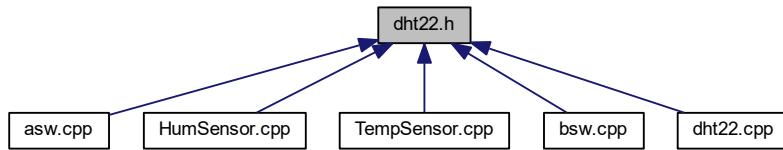
Pointer to `dht22` driver object

Definition at line 22 of file dht22.cpp.

## 5.14 dht22.h File Reference

DHT22 driver header file.

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



### Classes

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

### Variables

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

#### 5.14.1 Detailed Description

DHT22 driver header file.

##### Date

23 mars 2018

##### Author

nicls67

#### 5.14.2 Variable Documentation

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

`dht22* p_global_BSW_dht22`

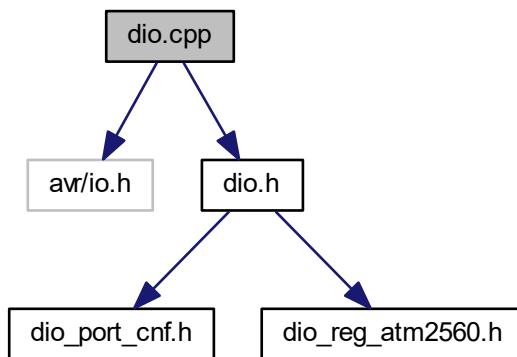
Pointer to [dht22](#) driver object

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

## 5.15 dio.cpp File Reference

DIO library.

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



### Variables

- [dio \\* p\\_global\\_BSW\\_dio](#)

#### 5.15.1 Detailed Description

DIO library.

##### Date

13 mars 2018

##### Author

nicls67

#### 5.15.2 Variable Documentation

### 5.15.2.1 p\_global\_BSW\_dio

`dio* p_global_BSW_dio`

Pointer to dio driver object

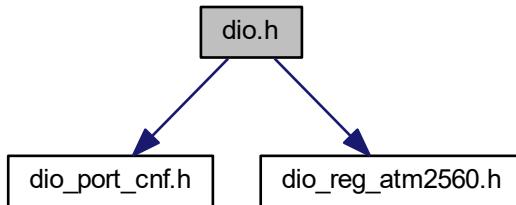
Definition at line 14 of file dio.cpp.

## 5.16 dio.h File Reference

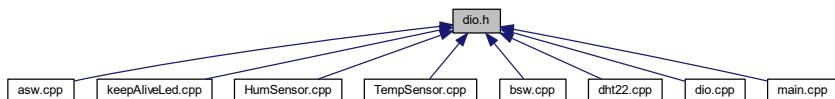
DIO library header file.

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

Include dependency graph for dio.h:



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



## Classes

- class `dio`  
*DIO class.*

## Macros

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

## Variables

- `dio * p_global_BSW_dio`

### 5.16.1 Detailed Description

DIO library header file.

#### Date

13 mars 2018

#### Author

nicls67

### 5.16.2 Macro Definition Documentation

#### 5.16.2.1 DECODE\_PIN

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

Macro used to extract pin index

Definition at line 20 of file dio.h.

#### 5.16.2.2 DECODE\_PORT

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

Macro used to extract port index

Definition at line 19 of file dio.h.

#### 5.16.2.3 ENCODE\_PORT

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

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

Definition at line 18 of file dio.h.

#### 5.16.2.4 PORT\_CNF\_IN

```
#define PORT_CNF_IN 0
```

Pin is configured as input

Definition at line 16 of file dio.h.

#### 5.16.2.5 PORT\_CNF\_OUT

```
#define PORT_CNF_OUT 1
```

Pin is configured as output

Definition at line 15 of file dio.h.

### 5.16.3 Variable Documentation

#### 5.16.3.1 p\_global\_BSW\_dio

```
dio* p_global_BSW_dio
```

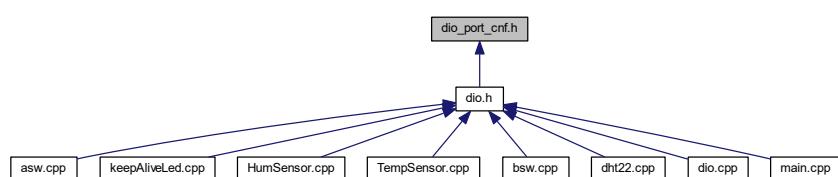
Pointer to dio driver object

Definition at line 14 of file dio.cpp.

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

Digital ports configuration file.

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



## Macros

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

### 5.17.1 Detailed Description

Digital ports configuration file.

#### Date

19 mars 2019

#### Author

nicls67

### 5.17.2 Macro Definition Documentation

#### 5.17.2.1 PORT\_A

`#define PORT_A 0`

PORTA index

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

#### 5.17.2.2 PORT\_B

`#define PORT_B 1`

PORTB index

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

### 5.17.2.3 PORT\_C

```
#define PORT_C 2
```

PORTC index

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

### 5.17.2.4 PORT\_D

```
#define PORT_D 3
```

PORTD index

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

### 5.17.2.5 PORTB\_CNF\_DDRC

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

Defines the configuration of DDRC register.

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

PB0 : N/A

PB1 : N/A

PB2 : N/A

PB3 : N/A

PB4 : IN

PB5 : N/A

PB6 : OUT

PB7 : OUT

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

### 5.17.2.6 PORTB\_CNF\_PORTB

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

Defines the configuration of PORTB register.

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

PB0 : N/A

PB1 : N/A

PB2 : N/A

PB3 : N/A

PB4 : Pull-up

PB5 : N/A

PB6 : HIGH

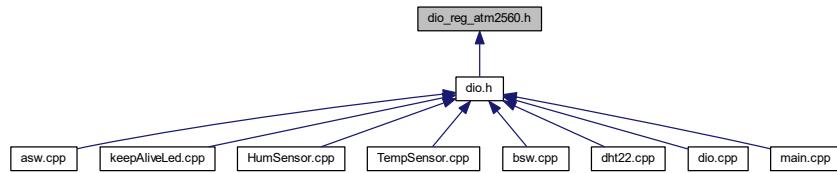
PB7 : LOW

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

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

Defines DIO register addresses for ATMEGA2560.

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



### Macros

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

### 5.18.1 Detailed Description

Defines DIO register addresses for ATMEGA2560.

#### Date

19 mars 2019

#### Author

nicls67

### 5.18.2 Macro Definition Documentation

### 5.18.2.1 DDRA\_PTR

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

Macro defining pointer to DDR A register

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

### 5.18.2.2 DDRB\_PTR

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

Macro defining pointer to DDR B register

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

### 5.18.2.3 DDRC\_PTR

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

Macro defining pointer to DDR C register

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

### 5.18.2.4 DDRD\_PTR

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

Macro defining pointer to DDR D register

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

### 5.18.2.5 PINA\_PTR

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

Macro defining pointer to PIN A register

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

### 5.18.2.6 PINB\_PTR

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

Macro defining pointer to PIN B register

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

### 5.18.2.7 PINC\_PTR

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

Macro defining pointer to PIN C register

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

### 5.18.2.8 PIND\_PTR

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

Macro defining pointer to PIN D register

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

### 5.18.2.9 PORTA\_PTR

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

Macro defining pointer to PORT A register

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

### 5.18.2.10 PORTB\_PTR

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

Macro defining pointer to PORT B register

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

### 5.18.2.11 PORTC\_PTR

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

Macro defining pointer to PORT C register

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

### 5.18.2.12 PORTD\_PTR

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

Macro defining pointer to PORT D register

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

## 5.19 DisplayInterface.cpp File Reference

Source code file for display services.

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

Include dependency graph for DisplayInterface.cpp:



### Variables

- [DisplayInterface \\* p\\_global\\_ASW\\_DisplayInterface](#)

### 5.19.1 Detailed Description

Source code file for display services.

#### Date

23 avr. 2019

#### Author

nicls67

### 5.19.2 Variable Documentation

#### 5.19.2.1 p\_global\_ASW\_DisplayInterface

```
DisplayInterface* p_global_ASW_DisplayInterface
```

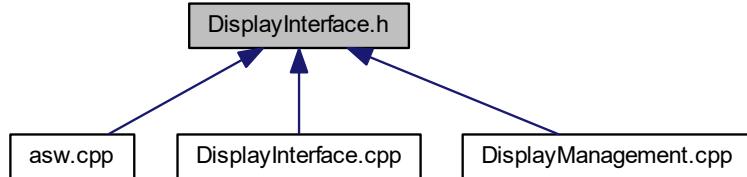
Pointer to [DisplayInterface](#) object

Definition at line 25 of file [DisplayInterface.cpp](#).

## 5.20 DisplayInterface.h File Reference

[DisplayInterface](#) class header file.

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



### Classes

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

### Macros

- #define [DISPLAY\\_LINE\\_SHIFT\\_PERIOD\\_MS](#) 500
- #define [DISPLAY\\_LINE\\_SHIFT\\_TEMPO\\_TIME](#) 6

## Enumerations

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

## Variables

- `DisplayInterface * p_global_ASW_DisplayInterface`

### 5.20.1 Detailed Description

`DisplayInterface` class header file.

Date

23 avr. 2019

Author

nicls67

### 5.20.2 Macro Definition Documentation

#### 5.20.2.1 DISPLAY\_LINE\_SHIFT\_PERIOD\_MS

```
#define DISPLAY_LINE_SHIFT_PERIOD_MS 500
```

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

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

#### 5.20.2.2 DISPLAY\_LINE\_SHIFT\_TEMPO\_TIME

```
#define DISPLAY_LINE_SHIFT_TEMPO_TIME 6
```

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

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

### 5.20.3 Enumeration Type Documentation

#### 5.20.3.1 T\_DisplayInterface\_LineAlignment

```
enum T_DisplayInterface_LineAlignment
```

Alignment mode for line display.

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

**Enumerator**

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

Definition at line 33 of file DisplayInterface.h.

### 5.20.3.2 T\_DisplayInterface\_LineDisplayMode

```
enum T_DisplayInterface_LineDisplayMode
```

Modes for line display.

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

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

**Enumerator**

NORMAL	
LINE_SHIFT	
GO_TO_NEXT_LINE	

Definition at line 20 of file DisplayInterface.h.

## 5.20.4 Variable Documentation

### 5.20.4.1 p\_global\_ASW\_DisplayInterface

```
DisplayInterface* p_global_ASW_DisplayInterface
```

Pointer to [DisplayInterface](#) object

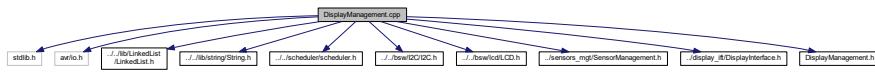
Definition at line 25 of file DisplayInterface.cpp.

## 5.21 DisplayManagement.cpp File Reference

Display management source file.

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

Include dependency graph for DisplayManagement.cpp:



### Variables

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

#### 5.21.1 Detailed Description

Display management source file.

##### Date

1 mai 2019

##### Author

nicls67

#### 5.21.2 Variable Documentation

##### 5.21.2.1 noSensorsDisplayString

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

**String** used in case sensors are deactivated

Definition at line 28 of file DisplayManagement.cpp.

### 5.21.2.2 p\_global\_ASW\_DisplayManagement

`DisplayManagement* p_global_ASW_DisplayManagement`

Pointer to `DisplayManagement` object

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

### 5.21.2.3 welcomeMessageString

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

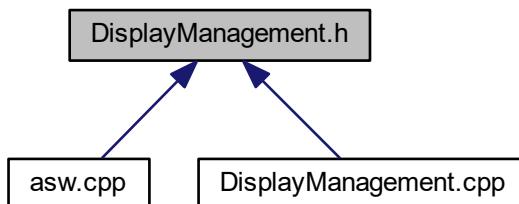
`String` displayed on the screen at startup

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

## 5.22 DisplayManagement.h File Reference

Display management class header file.

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



### Classes

- class `DisplayManagement`

*Display management class.*

### Macros

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

## Variables

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

### 5.22.1 Detailed Description

Display management class header file.

#### Date

1 mai 2019

#### Author

nicls67

### 5.22.2 Macro Definition Documentation

#### 5.22.2.1 DISPLAY\_MGT\_FIRST\_LINE\_SENSORS

```
#define DISPLAY_MGT_FIRST_LINE_SENSORS 0
```

Sensors data are displayed starting on line 0

Definition at line 18 of file DisplayManagement.h.

#### 5.22.2.2 DISPLAY\_MGT\_I2C\_BITRATE

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

I2C bus bitrate is 100 kHz

Definition at line 20 of file DisplayManagement.h.

#### 5.22.2.3 DISPLAY\_MGT\_LCD\_I2C\_ADDR

```
#define DISPLAY_MGT_LCD_I2C_ADDR 0x27
```

I2C address of the screen

Definition at line 13 of file DisplayManagement.h.

#### 5.22.2.4 DISPLAY\_MGT\_PERIOD\_TASK\_SENSOR

```
#define DISPLAY_MGT_PERIOD_TASK_SENSOR 1500
```

Display is updated every 1.5s

Definition at line 15 of file DisplayManagement.h.

#### 5.22.2.5 DISPLAY\_MGT\_PERIOD\_WELCOME\_MSG\_REMOVAL

```
#define DISPLAY_MGT_PERIOD_WELCOME_MSG_REMOVAL 5000
```

Time after which one the welcome message is removed

Definition at line 16 of file DisplayManagement.h.

### 5.22.3 Variable Documentation

#### 5.22.3.1 LCD\_init\_cnf

```
const T_LCD_conf_struct LCD_init_cnf
```

**Initial value:**

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

LCD configuration structure.

This structure defines the initial configuration of the LCD screen.

Definition at line 26 of file DisplayManagement.h.

#### 5.22.3.2 p\_global\_ASW\_DisplayManagement

```
DisplayManagement* p_global_ASW_DisplayManagement
```

Pointer to [DisplayManagement](#) object

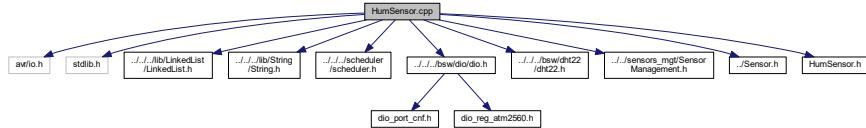
Definition at line 25 of file DisplayManagement.cpp.

## 5.23 HumSensor.cpp File Reference

Defines function of class [HumSensor](#).

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

Include dependency graph for HumSensor.cpp:



### Macros

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

#### 5.23.1 Detailed Description

Defines function of class [HumSensor](#).

##### Date

20 juin 2019

##### Author

nicls67

#### 5.23.2 Macro Definition Documentation

##### 5.23.2.1 DHT22\_PORT

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

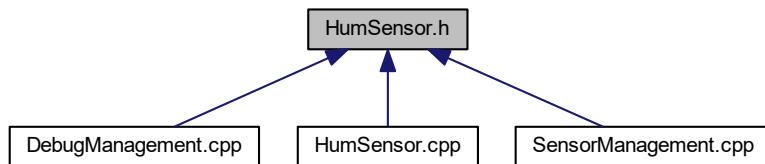
DHT22 is connected to port PB6

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

## 5.24 HumSensor.h File Reference

Class [HumSensor](#) header file.

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



### Classes

- class [HumSensor](#)

*Class for humidity sensor.*

#### 5.24.1 Detailed Description

Class [HumSensor](#) header file.

Date

20 juin 2019

Author

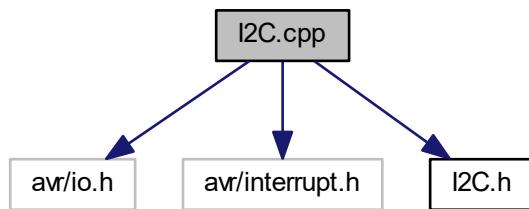
nicls67

## 5.25 I2C.cpp File Reference

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

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

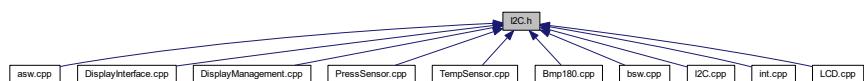
Pointer to [I2C](#) driver object

Definition at line 15 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 15 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.

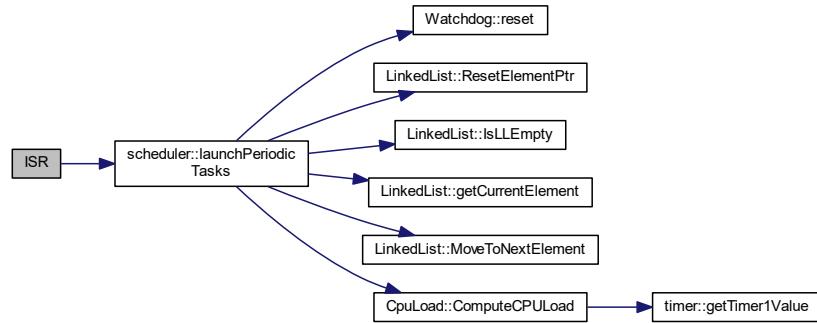
This function handles the interrupt raised by Timer #1. It wakes up the software every 500 ms to perform applications.

#### 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.

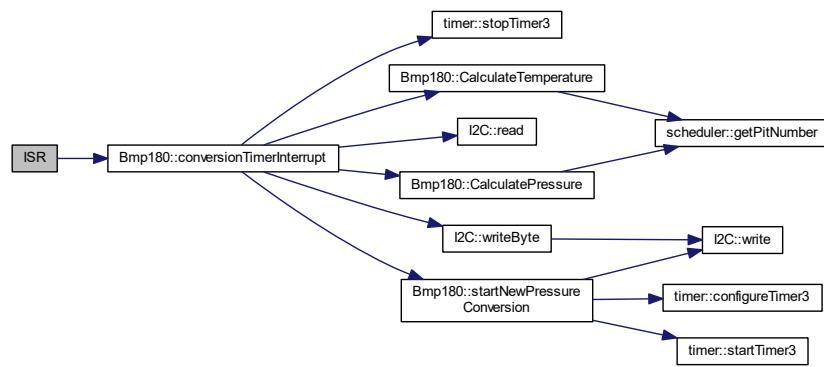
This function calls the end of conversion function of BMP180 driver.

**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.

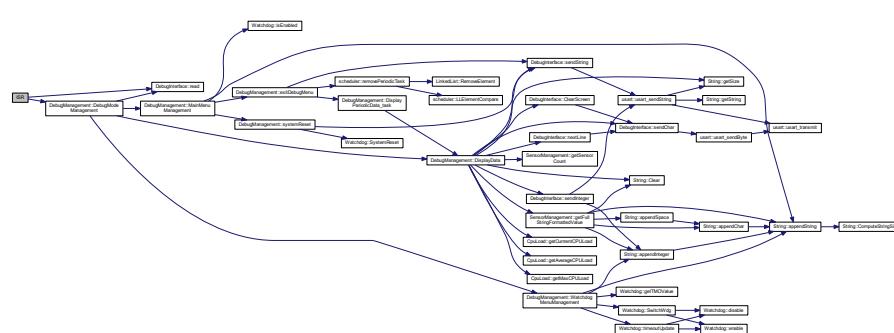
This function handles the interrupt raised when a frame has been received by USART. If debug mode mode is active, it calls debug mode management function. If inactive, it calls debug mode activation function if the received character is 'a'

**Returns**

Nothing

Definition at line 58 of file int.cpp.

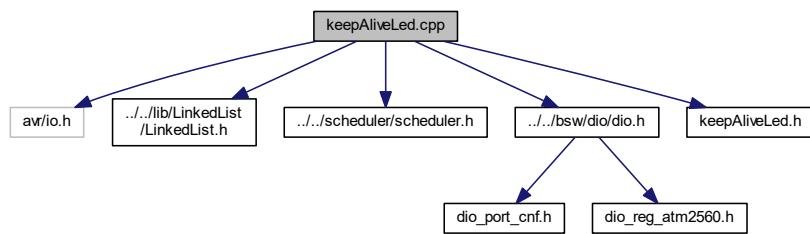
Here is the call graph for this function:



## 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`

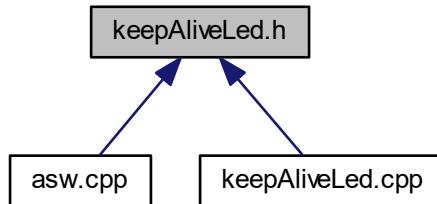
Pointer to [keepAliveLed](#) object

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

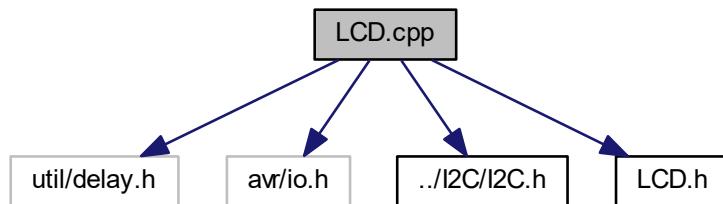
Definition at line 20 of file keepAliveLed.cpp.

## 5.30 LCD.cpp File Reference

[LCD](#) class source file.

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

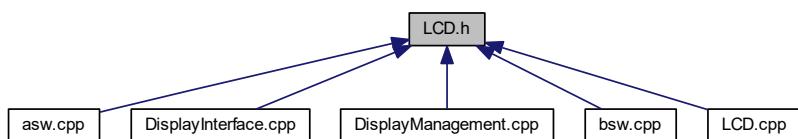
Pointer to `LCD` driver object

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

## 5.31 LCD.h File Reference

`LCD` class header file.

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



## Classes

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

## Macros

- #define EN\_PIN 2
- #define RW\_PIN 1
- #define RS\_PIN 0
- #define BACKLIGHT\_PIN 3
- #define LCD\_INST\_CLR\_DISPLAY\_BIT 0
- #define LCD\_INST\_FUNCTION\_SET 5
- #define LCD\_INST\_DISPLAY\_CTRL 3
- #define LCD\_INST\_ENTRY\_MODE\_SET 2
- #define LCD\_INST\_SET\_DDRAM\_ADDR 7
- #define LCD\_FCT\_SET\_FIELD\_DL 4
- #define LCD\_FCT\_SET\_FIELD\_N 3
- #define LCD\_FCT\_SET\_FIELD\_F 2
- #define LCD\_DISPLAY\_CTRL\_FIELD\_D 2
- #define LCD\_DISPLAY\_CTRL\_FIELD\_C 1
- #define LCD\_DISPLAY\_CTRL\_FIELD\_B 0
- #define LCD\_CNF\_SHIFT\_ID 1
- #define LCD\_CNF\_SHIFT\_SH 0
- #define LCD\_CNF\_ONE\_LINE 0
- #define LCD\_CNF\_TWO\_LINE 1
- #define LCD\_CNF\_FONT\_5\_8 0
- #define LCD\_CNF\_FONT\_5\_11 1
- #define LCD\_CNF\_DISPLAY\_ON 1
- #define LCD\_CNF\_DISPLAY\_OFF 0
- #define LCD\_CNF\_CURSOR\_ON 1
- #define LCD\_CNF\_CURSOR\_OFF 0
- #define LCD\_CNF\_CURSOR\_BLINK\_ON 1
- #define LCD\_CNF\_CURSOR\_BLINK\_OFF 0
- #define LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_RIGHT 1
- #define LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_LEFT 0
- #define LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT\_ON 1
- #define LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT\_OFF 0
- #define LCD\_CNF\_BACKLIGHT\_ON 1
- #define LCD\_CNF\_BACKLIGHT\_OFF 0
- #define LCD\_RAM\_1\_LINE\_MIN 0
- #define LCD\_RAM\_1\_LINE\_MAX 0x4F
- #define LCD\_RAM\_2\_LINES\_MIN\_1 0
- #define LCD\_RAM\_2\_LINES\_MAX\_1 0x27
- #define LCD\_RAM\_2\_LINES\_MIN\_2 0x40
- #define LCD\_RAM\_2\_LINES\_MAX\_2 0x67
- #define LCD\_WAIT\_CLR\_RETURN 1600
- #define LCD\_WAIT\_OTHER\_MODES 40
- #define LCD\_SIZE\_NB\_CHAR\_PER\_LINE 20
- #define LCD\_SIZE\_NB\_LINES 4

## Enumerations

- enum T\_LCD\_command {
 LCD\_CMD\_FUNCTION\_SET, LCD\_CMD\_CLEAR\_DISPLAY, LCD\_CMD\_DISPLAY\_CTRL, LCD\_CMD\_ENTRY\_MODE\_SET,
 LCD\_CMD\_SET\_DDRAM\_ADDR
 }
   
*LCD commands enumeration.*
- enum T\_LCD\_config\_mode { LCD\_MODE\_INSTRUCTION = 0, LCD\_MODE\_DATA = 1 }
   
*LCD modes enumeration.*
- enum T\_LCD\_ram\_area { LCD\_DATA\_DDRAM, LCD\_DATA\_CGRAM }
   
*Screen RAM definition.*

## Variables

- `LCD * p_global_BSW_lcd`

### 5.31.1 Detailed Description

`LCD` class header file.

#### Date

20 avr. 2019

#### Author

nicls67

### 5.31.2 Macro Definition Documentation

#### 5.31.2.1 BACKLIGHT\_PIN

```
#define BACKLIGHT_PIN 3
```

Backlight pin is on P3

Definition at line 17 of file LCD.h.

#### 5.31.2.2 EN\_PIN

```
#define EN_PIN 2
```

EN bit is on P2

Definition at line 14 of file LCD.h.

#### 5.31.2.3 LCD\_CNF\_BACKLIGHT\_OFF

```
#define LCD_CNF_BACKLIGHT_OFF 0
```

Backlight is disabled

Definition at line 70 of file LCD.h.

### 5.31.2.4 LCD\_CNF\_BACKLIGHT\_ON

```
#define LCD_CNF_BACKLIGHT_ON 1
```

Backlight is enabled

Definition at line 69 of file LCD.h.

### 5.31.2.5 LCD\_CNF\_CURSOR\_BLINK\_OFF

```
#define LCD_CNF_CURSOR_BLINK_OFF 0
```

Cursor blinking is off, bit is set to 0

Definition at line 58 of file LCD.h.

### 5.31.2.6 LCD\_CNF\_CURSOR\_BLINK\_ON

```
#define LCD_CNF_CURSOR_BLINK_ON 1
```

Cursor blinking is on, bit is set to 1

Definition at line 57 of file LCD.h.

### 5.31.2.7 LCD\_CNF\_CURSOR\_OFF

```
#define LCD_CNF_CURSOR_OFF 0
```

Cursor is off, bit is set to 0

Definition at line 54 of file LCD.h.

### 5.31.2.8 LCD\_CNF\_CURSOR\_ON

```
#define LCD_CNF_CURSOR_ON 1
```

Cursor is on, bit is set to 1

Definition at line 53 of file LCD.h.

### 5.31.2.9 LCD\_CNF\_DISPLAY\_OFF

```
#define LCD_CNF_DISPLAY_OFF 0
```

Display is off, bit is set to 0

Definition at line 50 of file LCD.h.

### 5.31.2.10 LCD\_CNF\_DISPLAY\_ON

```
#define LCD_CNF_DISPLAY_ON 1
```

Display is on, bit is set to 1

Definition at line 49 of file LCD.h.

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

```
#define LCD_CNF_ENTRY_MODE_DIRECTION_LEFT 0
```

Direction of shift is left, bit is set to 0

Definition at line 62 of file LCD.h.

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

```
#define LCD_CNF_ENTRY_MODE_DIRECTION_RIGHT 1
```

Direction of shift is right, bit is set to 1

Definition at line 61 of file LCD.h.

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

```
#define LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_OFF 0
```

Display shift is not performed, bit is set to 0

Definition at line 66 of file LCD.h.

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

```
#define LCD_CNF_ENTRY_MODE_DISPLAY_SHIFT_ON 1
```

Display shift is performed, bit is set to 1

Definition at line 65 of file LCD.h.

### 5.31.2.15 LCD\_CNF\_FONT\_5\_11

```
#define LCD_CNF_FONT_5_11 1
```

Two-line configuration, bit is set to 1

Definition at line 46 of file LCD.h.

### 5.31.2.16 LCD\_CNF\_FONT\_5\_8

```
#define LCD_CNF_FONT_5_8 0
```

One-line configuration, bit is set to 0

Definition at line 45 of file LCD.h.

### 5.31.2.17 LCD\_CNF\_ONE\_LINE

```
#define LCD_CNF_ONE_LINE 0
```

One-line configuration, bit is set to 0

Definition at line 41 of file LCD.h.

### 5.31.2.18 LCD\_CNF\_SHIFT\_ID

```
#define LCD_CNF_SHIFT_ID 1
```

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

Definition at line 37 of file LCD.h.

**5.31.2.19 LCD\_CNF\_SHIFT\_SH**

```
#define LCD_CNF_SHIFT_SH 0
```

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

Definition at line 38 of file LCD.h.

**5.31.2.20 LCD\_CNF\_TWO\_LINE**

```
#define LCD_CNF_TWO_LINE 1
```

Two-line configuration, bit is set to 1

Definition at line 42 of file LCD.h.

**5.31.2.21 LCD\_DISPLAY\_CTRL\_FIELD\_B**

```
#define LCD_DISPLAY_CTRL_FIELD_B 0
```

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

Definition at line 34 of file LCD.h.

**5.31.2.22 LCD\_DISPLAY\_CTRL\_FIELD\_C**

```
#define LCD_DISPLAY_CTRL_FIELD_C 1
```

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

Definition at line 33 of file LCD.h.

**5.31.2.23 LCD\_DISPLAY\_CTRL\_FIELD\_D**

```
#define LCD_DISPLAY_CTRL_FIELD_D 2
```

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

Definition at line 32 of file LCD.h.

### 5.31.2.24 LCD\_FCT\_SET\_FIELD\_DL

```
#define LCD_FCT_SET_FIELD_DL 4
```

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

Definition at line 27 of file LCD.h.

### 5.31.2.25 LCD\_FCT\_SET\_FIELD\_F

```
#define LCD_FCT_SET_FIELD_F 2
```

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

Definition at line 29 of file LCD.h.

### 5.31.2.26 LCD\_FCT\_SET\_FIELD\_N

```
#define LCD_FCT_SET_FIELD_N 3
```

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

Definition at line 28 of file LCD.h.

### 5.31.2.27 LCD\_INST\_CLR\_DISPLAY\_BIT

```
#define LCD_INST_CLR_DISPLAY_BIT 0
```

Instruction bit for "clear display" is DB0

Definition at line 20 of file LCD.h.

### 5.31.2.28 LCD\_INST\_DISPLAY\_CTRL

```
#define LCD_INST_DISPLAY_CTRL 3
```

Instruction bit for "display control" is DB3

Definition at line 22 of file LCD.h.

**5.31.2.29 LCD\_INST\_ENTRY\_MODE\_SET**

```
#define LCD_INST_ENTRY_MODE_SET 2
```

Instruction bit for "entry mode" is DB2

Definition at line 23 of file LCD.h.

**5.31.2.30 LCD\_INST\_FUNCTION\_SET**

```
#define LCD_INST_FUNCTION_SET 5
```

Instruction bit for "function set" is DB5

Definition at line 21 of file LCD.h.

**5.31.2.31 LCD\_INST\_SET\_DDRAM\_ADDR**

```
#define LCD_INST_SET_DDRAM_ADDR 7
```

Instruction bit for "set DDRAM address" is DB7

Definition at line 24 of file LCD.h.

**5.31.2.32 LCD\_RAM\_1\_LINE\_MAX**

```
#define LCD_RAM_1_LINE_MAX 0x4F
```

Maximum address value in 1-line mode

Definition at line 74 of file LCD.h.

**5.31.2.33 LCD\_RAM\_1\_LINE\_MIN**

```
#define LCD_RAM_1_LINE_MIN 0
```

Minimum address value in 1-line mode

Definition at line 73 of file LCD.h.

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

```
#define LCD_RAM_2_LINES_MAX_1 0x27
```

Maximum address value in 2-lines mode for line 1

Definition at line 76 of file LCD.h.

### 5.31.2.35 LCD\_RAM\_2\_LINES\_MAX\_2

```
#define LCD_RAM_2_LINES_MAX_2 0x67
```

Maximum address value in 2-lines mode for line 2

Definition at line 78 of file LCD.h.

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

```
#define LCD_RAM_2_LINES_MIN_1 0
```

Minimum address value in 2-lines mode for line 1

Definition at line 75 of file LCD.h.

### 5.31.2.37 LCD\_RAM\_2\_LINES\_MIN\_2

```
#define LCD_RAM_2_LINES_MIN_2 0x40
```

Minimum address value in 2-lines mode for line 2

Definition at line 77 of file LCD.h.

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

```
#define LCD_SIZE_NB_CHAR_PER_LINE 20
```

LCD screen has 20 characters per line

Definition at line 85 of file LCD.h.

### 5.31.2.39 LCD\_SIZE\_NB\_LINES

```
#define LCD_SIZE_NB_LINES 4
```

LCD screen has 4 lines

Definition at line 86 of file LCD.h.

### 5.31.2.40 LCD\_WAIT\_CLR\_RETURN

```
#define LCD_WAIT_CLR_RETURN 1600
```

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

Definition at line 81 of file LCD.h.

### 5.31.2.41 LCD\_WAIT\_OTHER\_MODES

```
#define LCD_WAIT_OTHER_MODES 40
```

Waiting time after all other modes is at least 38 us

Definition at line 82 of file LCD.h.

### 5.31.2.42 RS\_PIN

```
#define RS_PIN 0
```

RS pin is on P0

Definition at line 16 of file LCD.h.

### 5.31.2.43 RW\_PIN

```
#define RW_PIN 1
```

RW pin is on P1

Definition at line 15 of file LCD.h.

## 5.31.3 Enumeration Type Documentation

### 5.31.3.1 T\_LCD\_command

```
enum T_LCD_command
```

LCD commands enumeration.

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

**Enumerator**

LCD_CMDFUNCTION_SET	
LCD_CMDCLEAR_DISPLAY	
LCD_CMDDISPLAY_CTRL	
LCD_CMDENTRY_MODE_SET	
LCD_CMDSET_DDRAM_ADDR	

Definition at line 93 of file LCD.h.

**5.31.3.2 T\_LCD\_config\_mode**

```
enum T_LCD_config_mode
```

LCD modes enumeration.

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

**Enumerator**

LCD_MODEINSTRUCTION	
LCD_MODEDATA	

Definition at line 107 of file LCD.h.

**5.31.3.3 T\_LCD\_ram\_area**

```
enum T_LCD_ram_area
```

Screen RAM definition.

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

**Enumerator**

LCD_DATA_DDRAM	
LCD_DATA_CGRAM	

Definition at line 118 of file LCD.h.

**5.31.4 Variable Documentation**

#### **5.31.4.1 p\_global\_BSW\_lcd**

LCD\* p\_global\_BSW\_lcd

Pointer to [LCD](#) driver object

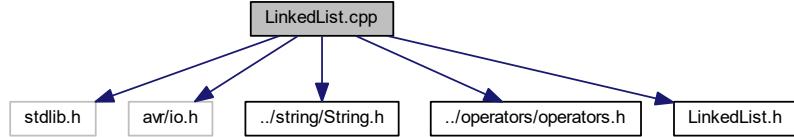
Definition at line 16 of file LCD.cpp.

## 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

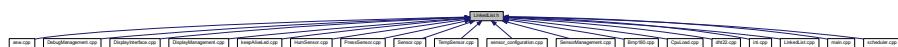
## Author

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

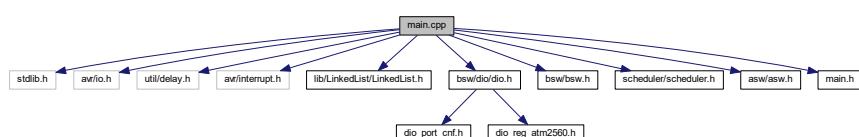
Definition at line 14 of file `LinkedList.h`.

## 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()

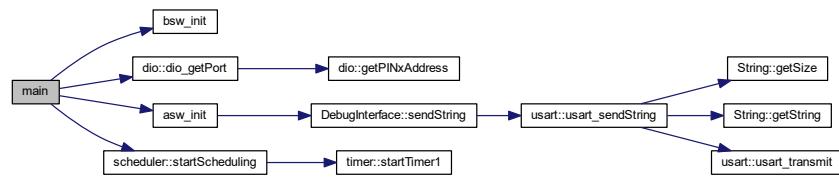
```
int main (
    void )
```

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
```

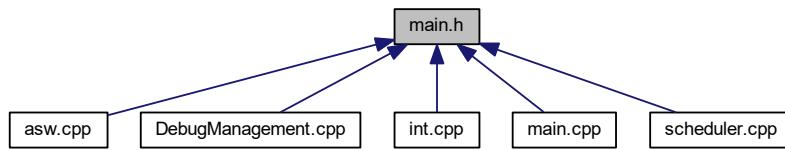
Flag indicating if the debug mode is activated or not

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
```

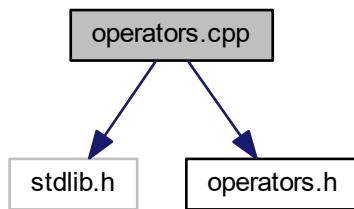
Flag indicating if the debug mode is activated or not

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

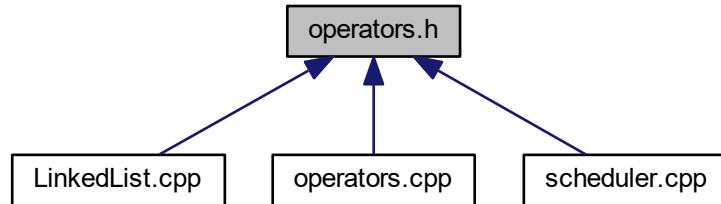
Definition at line 13 of file operators.cpp.

## 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**

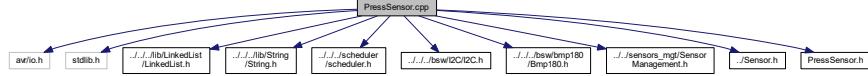
Pointer to the start of allocated memory zone

Definition at line 13 of file operators.cpp.

**5.38 PressSensor.cpp File Reference**Defines function of class [PressSensor](#).

```
#include <avr/io.h>
#include <stdlib.h>
#include "../../lib/LinkedList/LinkedList.h"
#include "../../lib/String/String.h"
#include "../../scheduler/scheduler.h"
#include "../../bsw/I2C/I2C.h"
#include "../../bsw/bmp180/Bmp180.h"
#include "../../sensors_mgt/SensorManagement.h"
#include "../Sensor.h"
#include "PressSensor.h"
```

Include dependency graph for PressSensor.cpp:



### 5.38.1 Detailed Description

Defines function of class [PressSensor](#).

Date

6 aout 2019

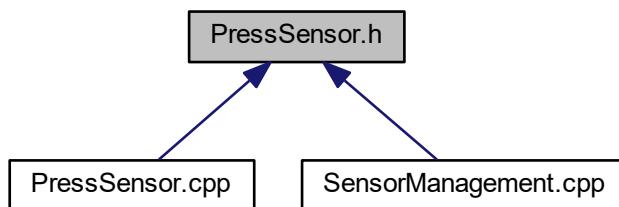
Author

nicls67

## 5.39 PressSensor.h File Reference

Class [PressSensor](#) header file.

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



### Classes

- class [PressSensor](#)  
*Class for pressure sensor.*

### 5.39.1 Detailed Description

Class [PressSensor](#) header file.

Date

6 aout 2019

Author

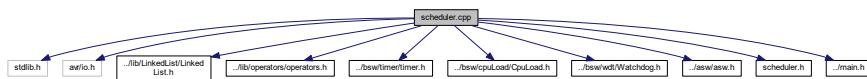
nicls67

## 5.40 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`

### 5.40.1 Detailed Description

Defines scheduler class.

#### Date

16 mars 2018

#### Author

nicls67

### 5.40.2 Variable Documentation

#### 5.40.2.1 `p_global_scheduler`

`scheduler* p_global_scheduler`

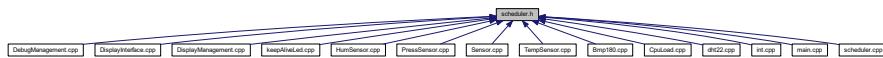
Pointer to scheduler object

Definition at line 27 of file scheduler.cpp.

## 5.41 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](#)

#### 5.41.1 Detailed Description

Scheduler class header file.

##### Date

16 mars 2018

##### Author

nicls67

#### 5.41.2 Macro Definition Documentation

#### 5.41.2.1 PRESCALER\_PERIODIC\_TIMER

```
#define PRESCALER_PERIODIC_TIMER 256
```

Value of prescaler to use for periodic timer

Definition at line 16 of file scheduler.h.

#### 5.41.2.2 SW\_PERIOD\_MS

```
#define SW_PERIOD_MS 500
```

Software period, used to define periodic timer interrupt

Definition at line 15 of file scheduler.h.

#### 5.41.2.3 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.

### 5.41.3 Typedef Documentation

#### 5.41.3.1 TaskPtr\_t

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

Type defining a pointer to function.

Definition at line 22 of file scheduler.h.

### 5.41.4 Variable Documentation

#### 5.41.4.1 p\_global\_scheduler

`scheduler* p_global_scheduler`

Pointer to scheduler object

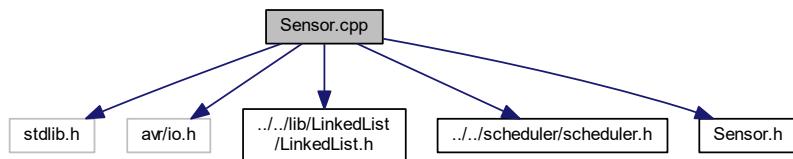
Definition at line 27 of file scheduler.cpp.

## 5.42 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`

#### 5.42.1 Detailed Description

[Sensor](#) class source code file.

##### Date

20 juin 2019

##### Author

nicls67

#### 5.42.2 Macro Definition Documentation

### 5.42.2.1 TASK\_PERIOD\_DEFAULT

```
#define TASK_PERIOD_DEFAULT 1000
```

Default sensor task period : 1s

Definition at line 18 of file Sensor.cpp.

### 5.42.2.2 VALIDITY\_TIMEOUT\_MS\_DEFAULT

```
#define VALIDITY_TIMEOUT_MS_DEFAULT 30000
```

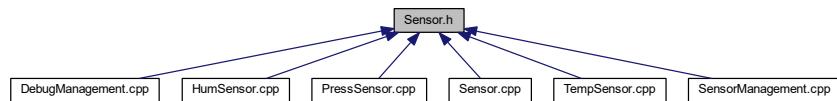
Sensor data are declared invalid after 30s

Definition at line 19 of file Sensor.cpp.

## 5.43 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.*

### 5.43.1 Detailed Description

Sensor class header file.

#### Date

20 juin 2019

#### Author

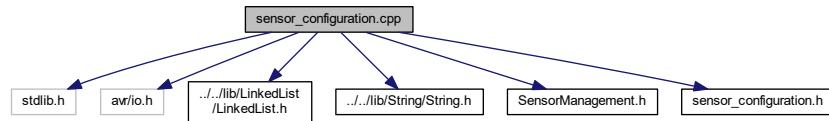
nicls67

## 5.44 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 [3]`  
*Sensor configuration table.*

#### 5.44.1 Detailed Description

[Sensor](#) configuration file.

##### Date

22 juin 2019

##### Author

nicls67

#### 5.44.2 Macro Definition Documentation

#### 5.44.2.1 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.

#### 5.44.2.2 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.

### 5.44.3 Variable Documentation

#### 5.44.3.1 SensorManagement\_Sensor\_Config\_list

[T\\_SensorManagement\\_Sensor\\_Config](#) SensorManagement\_Sensor\_Config\_list[3]

**Initial value:**

```
=
{
    {
        TEMPERATURE,
        SENSOR_MGT_CNF_DEFAULT_PERIOD,
        SENSOR_MGT_CNF_DEFAULT_TMO,
        (uint8_t*)"T",
        (uint8_t*)"degC"
    },
    {
        HUMIDITY,
        SENSOR_MGT_CNF_DEFAULT_PERIOD,
        SENSOR_MGT_CNF_DEFAULT_TMO,
        (uint8_t*)"H",
        (uint8_t*)"%"
    },
    {
        PRESSURE,
        SENSOR_MGT_CNF_DEFAULT_PERIOD,
        SENSOR_MGT_CNF_DEFAULT_TMO,
        (uint8_t*)"P",
        (uint8_t*)"hPa"
    }
}
```

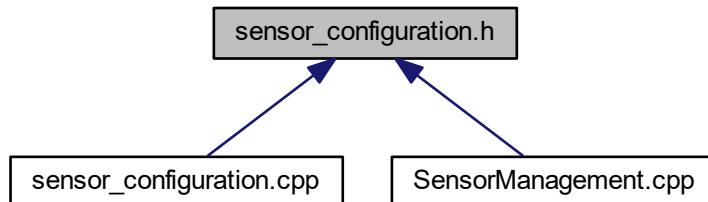
[Sensor](#) configuration table.

Definition at line 25 of file sensor\_configuration.cpp.

## 5.45 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 [3]`  
*Sensor configuration table.*

#### 5.45.1 Detailed Description

Sensors configuration header file.

##### Date

22 juin 2019

##### Author

nicls67

#### 5.45.2 Variable Documentation

### 5.45.2.1 SensorManagement\_Sensor\_Config\_list

```
T_SensorManagement_Sensor_Config SensorManagement_Sensor_Config_list[3]
```

**Sensor** configuration table.

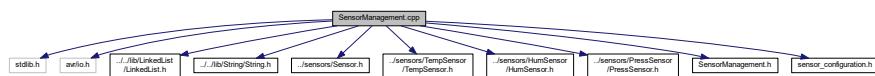
Definition at line 25 of file sensor\_configuration.cpp.

## 5.46 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 "../sensors/PressSensor/PressSensor.h"
#include "SensorManagement.h"
#include "sensor_configuration.h"
```

Include dependency graph for SensorManagement.cpp:



## Variables

- **SensorManagement \* p\_global\_ASW\_SensorManagement**

### 5.46.1 Detailed Description

**SensorManagement** class source code file.

#### Date

22 juin 2019

#### Author

nicls67

### 5.46.2 Variable Documentation

### 5.46.2.1 p\_global\_ASW\_SensorManagement

`SensorManagement* p_global_ASW_SensorManagement`

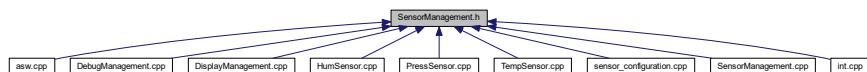
Pointer to the `SensorManagement` object

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

## 5.47 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`, `PRESSURE` }
- Sensor type enumeration.*

### Variables

- `SensorManagement* p_global_ASW_SensorManagement`

### 5.47.1 Detailed Description

`SensorManagement` class header file.

#### Date

22 juin 2019

#### Author

nicls67

## 5.47.2 Enumeration Type Documentation

### 5.47.2.1 T\_SensorManagement\_Sensor\_Type

`enum T_SensorManagement_Sensor_Type`

*Sensor type enumeration.*

This enumeration defines all types of sensors available.

Enumerator

TEMPERATURE	
HUMIDITY	
PRESSURE	

Definition at line 17 of file SensorManagement.h.

### 5.47.3 Variable Documentation

#### 5.47.3.1 p\_global\_ASW\_SensorManagement

`SensorManagement* p_global_ASW_SensorManagement`

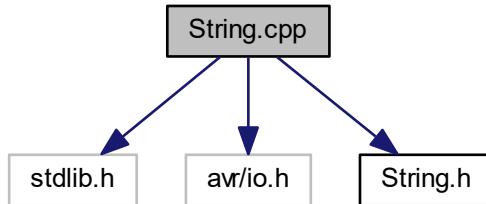
Pointer to the [SensorManagement](#) object

Definition at line 24 of file SensorManagement.cpp.

## 5.48 String.cpp File Reference

[String](#) class source file.

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



### 5.48.1 Detailed Description

[String](#) class source file.

Date

2 mai 2019

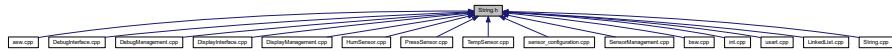
Author

nicls67

## 5.49 String.h File Reference

## [String](#) class header file.

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



## Classes

- class `String`

### **5.49.1 Detailed Description**

## [String](#) class header file.

Date

2 mai 2019

## Author

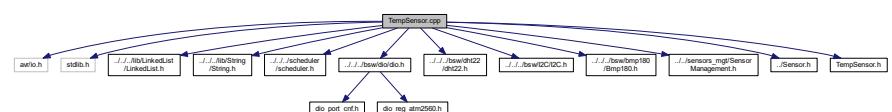
nicls67

## 5.50 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`

### 5.50.1 Detailed Description

Defines function of class [TempSensor](#).

#### Date

23 mars 2018

#### Author

nicls67

### 5.50.2 Macro Definition Documentation

#### 5.50.2.1 DHT22\_PORT

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

DHT22 is connected to port PB6

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

#### 5.50.2.2 TEMP\_SENSOR\_VALUE\_ACCEPTABLE\_DIFFERENCE

```
#define TEMP_SENSOR_VALUE_ACCEPTABLE_DIFFERENCE 10
```

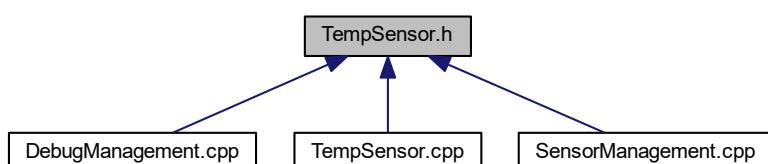
Sensors value can have a difference of 1 degree before becoming invalid

Definition at line 27 of file [TempSensor.cpp](#).

## 5.51 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.*

### 5.51.1 Detailed Description

Class [TempSensor](#) header file.

Date

23 mars 2018

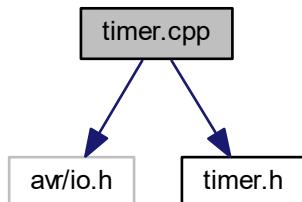
Author

nicls67

## 5.52 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](#)

### 5.52.1 Detailed Description

Defines function for class timer.

Date

15 mars 2018

Author

nicls67

### 5.52.2 Variable Documentation

#### 5.52.2.1 p\_global\_BSW\_timer

```
timer* p_global_BSW_timer
```

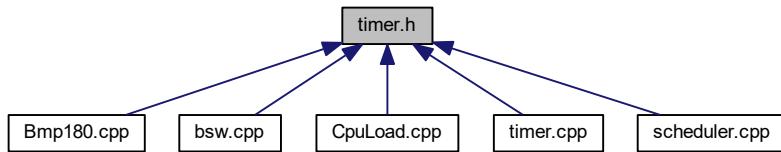
Pointer to timer driver object

Definition at line 13 of file timer.cpp.

## 5.53 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](#)

### 5.53.1 Detailed Description

Timer class header file.

#### Date

15 mars 2018

#### Author

nicls67

### 5.53.2 Variable Documentation

#### 5.53.2.1 p\_global\_BSW\_timer

```
timer* p_global_BSW_timer
```

Pointer to timer driver object

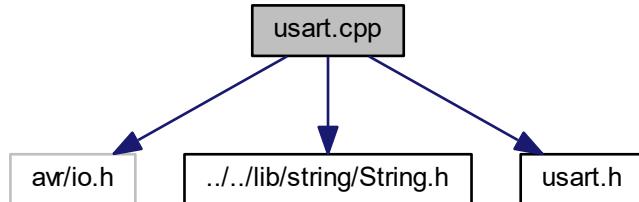
Definition at line 13 of file timer.cpp.

## 5.54 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

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

### 5.54.1 Detailed Description

BSW library for USART.

#### Date

13 mars 2018

#### Author

nicls67

### 5.54.2 Variable Documentation

#### 5.54.2.1 p\_global\_BSW\_usart

```
uart* p_global_BSW_usart
```

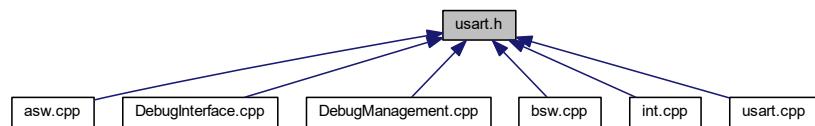
Pointer to usart driver object

Definition at line 16 of file usart.cpp.

## 5.55 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](#)

#### 5.55.1 Detailed Description

Header file for USART library.

#### Date

13 mars 2018

#### Author

nicls67

## 5.55.2 Variable Documentation

### 5.55.2.1 p\_global\_BSW\_usart

```
usart* p_global_BSW_usart
```

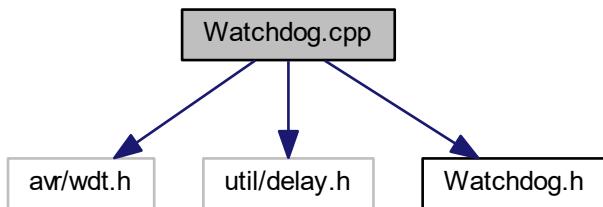
Pointer to usart driver object

Definition at line 16 of file usart.cpp.

## 5.56 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](#)

### 5.56.1 Detailed Description

Class [Watchdog](#) source code file.

Date

6 juin 2019

Author

nicls67

### 5.56.2 Macro Definition Documentation

#### 5.56.2.1 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.

### 5.56.3 Variable Documentation

#### 5.56.3.1 p\_global\_BSW\_wdg

```
Watchdog* p_global_BSW_wdg
```

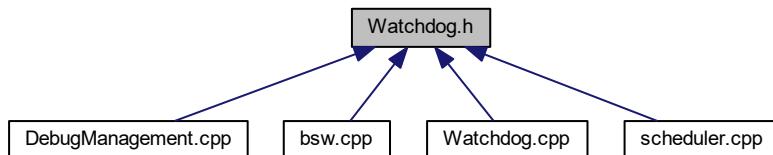
Pointer to [Watchdog](#) driver object

Definition at line 17 of file Watchdog.cpp.

## 5.57 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](#)

### 5.57.1 Detailed Description

Class [Watchdog](#) header file.

#### Date

6 juin 2019

#### Author

nicls67

### 5.57.2 Macro Definition Documentation

#### 5.57.2.1 WDG\_TMO\_120MS

```
#define WDG_TMO_120MS WDTO_120MS
```

Timeout value is 120 ms

Definition at line 19 of file Watchdog.h.

### 5.57.2.2 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.

### 5.57.2.3 WDG\_TMO\_1S

```
#define WDG_TMO_1S WDTO_1S
```

Timeout value is 1 s

Definition at line 22 of file Watchdog.h.

### 5.57.2.4 WDG\_TMO\_250MS

```
#define WDG_TMO_250MS WDTO_250MS
```

Timeout value is 250 ms

Definition at line 20 of file Watchdog.h.

### 5.57.2.5 WDG\_TMO\_2S

```
#define WDG_TMO_2S WDTO_2S
```

Timeout value is 2 s

Definition at line 23 of file Watchdog.h.

### 5.57.2.6 WDG\_TMO\_30MS

```
#define WDG_TMO_30MS WDTO_30MS
```

Timeout value is 30 ms

Definition at line 17 of file Watchdog.h.

### 5.57.2.7 WDG\_TMO\_4S

```
#define WDG_TMO_4S WDTO_4S
```

Timeout value is 4 s

Definition at line 24 of file Watchdog.h.

### 5.57.2.8 WDG\_TMO\_500MS

```
#define WDG_TMO_500MS WDTO_500MS
```

Timeout value is 500 ms

Definition at line 21 of file Watchdog.h.

### 5.57.2.9 WDG\_TMO\_60MS

```
#define WDG_TMO_60MS WDTO_60MS
```

Timeout value is 60 ms

Definition at line 18 of file Watchdog.h.

### 5.57.2.10 WDG\_TMO\_8S

```
#define WDG_TMO_8S WDTO_8S
```

Timeout value is 8 s

Definition at line 25 of file Watchdog.h.

## 5.57.3 Variable Documentation

### 5.57.3.1 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, 117

~String  
    String, 151

AC1  
    Bmp180::T\_BMP180\_calib\_data, 161

AC2  
    Bmp180::T\_BMP180\_calib\_data, 161

AC3  
    Bmp180::T\_BMP180\_calib\_data, 161

AC4  
    Bmp180::T\_BMP180\_calib\_data, 161

AC5  
    Bmp180::T\_BMP180\_calib\_data, 161

AC6  
    Bmp180::T\_BMP180\_calib\_data, 161

ASW\_init\_cnf  
    main.cpp, 266  
    main.h, 267

ActivatePressureConversion  
    Bmp180, 12

ActivateTemperatureConversion  
    Bmp180, 12

addPeriodicTask  
    scheduler, 129

alignment  
    T\_display\_data, 164

appendBool  
    String, 152

appendChar  
    String, 152

appendInteger  
    String, 153

appendSpace  
    String, 154

appendString  
    String, 155

asw.cpp, 197  
    asw\_init, 198

asw.h, 199  
    asw\_init, 199

asw\_init  
    asw.cpp, 198  
    asw.h, 199

AttachNewElement  
    LinkedList, 118

avg\_load  
    CpuLoad, 31

B1  
    Bmp180::T\_BMP180\_calib\_data, 162

B2  
    Bmp180::T\_BMP180\_calib\_data, 162

B5\_mem  
    Bmp180, 26

BACKLIGHT\_PIN  
    LCD.h, 253

BMP180\_CHIP\_ID\_CALIB\_EEP\_START\_ADDR  
    Bmp180.h, 203

BMP180\_CHIP\_ID\_EEP\_ADDR  
    Bmp180.h, 203

BMP180\_CHIP\_ID\_EXPECTED  
    Bmp180.h, 203

BMP180\_CTRL\_MEAS\_EEP\_ADDR  
    Bmp180.h, 203

BMP180\_CTRL\_MEAS\_START\_PRESS\_CONV\_OS←  
    S0  
    Bmp180.h, 203

BMP180\_CTRL\_MEAS\_START\_TEMP\_CONV  
    Bmp180.h, 203

BMP180\_I2C\_ADDR  
    Bmp180.h, 204

BMP180\_I2C\_BITRATE  
    Bmp180.h, 204

BMP180\_MONITORING\_DEFAULT\_PERIOD  
    Bmp180.h, 204

BMP180\_OUT\_REG\_LSB EEPROM\_ADDR  
    Bmp180.h, 204

BMP180\_OUT\_REG\_MSB EEPROM\_ADDR  
    Bmp180.h, 204

BMP180\_PRESS\_MEAS\_OSS0\_TIMER\_CTC\_VALUE  
    Bmp180.h, 205

BMP180\_PRESS\_MEAS\_OSS0\_WAITING\_TIME  
    Bmp180.h, 205

BMP180\_TEMP\_MEAS\_TIMER\_CTC\_VALUE  
    Bmp180.h, 205

BMP180\_TEMP\_MEAS\_WAITING\_TIME  
    Bmp180.h, 205

BMP180\_TIMER\_PRESCALER\_VALUE  
    Bmp180.h, 205

backlight\_en  
    T\_LCD\_conf\_struct, 168

backlight\_enable  
    LCD, 113

BaudRate  
    uart, 188

bitrate  
    I2C, 96

blinkLed\_task  
 keepAliveLed, 97

Bmp180, 9  
 ActivatePressureConversion, 12  
 ActivateTemperatureConversion, 12  
 B5\_mem, 26  
 Bmp180, 11  
 Bmp180Monitoring\_Task, 13  
 CalculatePressure, 14  
 CalculateTemperature, 15  
 calibration\_data, 26  
 chip\_id, 26  
 conversionTimerInterrupt, 16  
 getMonitoringTaskPeriod, 17  
 getPressureValue, 17  
 getStatus, 18  
 getTemperatureValue, 18  
 i2c\_drv\_ptr, 26  
 isPressConvActivated, 26  
 isPressConversionActivated, 19  
 isTempConvActivated, 26  
 isTempConversionActivated, 19  
 pressure\_value, 27  
 PressureMonitoring, 20  
 readCalibData, 20  
 readChipID, 21  
 startNewPressureConversion, 22  
 startNewTemperatureConversion, 23  
 status, 27  
 StopPressureConversion, 24  
 StopTemperatureConversion, 24  
 task\_period, 27  
 temperature\_value, 27  
 TemperatureMonitoring, 25

Bmp180.cpp, 200  
 p\_global\_BSW\_bmp180, 201

Bmp180.h, 201  
 BMP180\_CHIP\_ID\_CALIB\_EEP\_START\_ADDR, 203  
 BMP180\_CHIP\_ID\_EEP\_ADDR, 203  
 BMP180\_CHIP\_ID\_EXPECTED, 203  
 BMP180\_CTRL\_MEAS\_EEP\_ADDR, 203  
 BMP180\_CTRL\_MEAS\_START\_PRESS\_CON←  
     V\_OSS0, 203  
 BMP180\_CTRL\_MEAS\_START\_TEMP\_CONV, 203  
 BMP180\_I2C\_ADDR, 204  
 BMP180\_I2C\_BITRATE, 204  
 BMP180\_MONITORING\_DEFAULT\_PERIOD, 204  
 BMP180\_OUT\_REG\_LSB EEPROM\_ADDR, 204  
 BMP180\_OUT\_REG\_MSB EEPROM\_ADDR, 204  
 BMP180\_PRESS\_MEAS\_OSS0\_TIMER\_CTC←  
     VALUE, 205  
 BMP180\_PRESS\_MEAS\_OSS0\_WAITING\_TIME, 205  
 BMP180\_TEMP\_MEAS\_TIMER\_CTC\_VALUE, 205

BMP180\_TEMP\_MEAS\_WAITING\_TIME, 205  
 BMP180\_TIMER\_PRESCALER\_VALUE, 205  
 p\_global\_BSW\_bmp180, 206  
 T\_BMP180\_status, 206

Bmp180::T\_BMP180\_calib\_data, 160  
 AC1, 161  
 AC2, 161  
 AC3, 161  
 AC4, 161  
 AC5, 161  
 AC6, 161  
 B1, 162  
 B2, 162  
 MB, 162  
 MC, 162  
 MD, 162

Bmp180::T\_BMP180\_measurement\_data, 163  
 ready, 163  
 ts, 163  
 value, 163

Bmp180Monitoring\_Task  
 Bmp180, 13

bsw.cpp, 206  
 bsw\_init, 207

bsw.h, 208  
 bsw\_init, 209

bsw\_init  
 bsw.cpp, 207  
 bsw.h, 209

CalculatePressure  
 Bmp180, 14

CalculateTemperature  
 Bmp180, 15

calibration\_data  
 Bmp180, 26

chip\_id  
 Bmp180, 26

Clear  
 String, 156

ClearFullScreen  
 DisplayInterface, 71

ClearLine  
 DisplayInterface, 72

ClearScreen  
 DebugInterface, 35

ClearStringInDataStruct  
 DisplayInterface, 73

cnfCursorBlink  
 LCD, 113

cnfCursorOnOff  
 LCD, 113

cnfDisplayOnOff  
 LCD, 113

cnfEntryModeDir  
 LCD, 113

cnfEntryModeShift  
 LCD, 113

cnfFontType

LCD, 114  
cnfI2C\_addr  
    LCD, 114  
cnfLineNumber  
    LCD, 114  
command  
    LCD, 101  
CompareFctPtr\_t  
    LinkedList.h, 264  
ComputeCPUload  
    CpuLoad, 29  
ComputeStringSize  
    String, 156  
ConfigureBacklight  
    LCD, 102  
ConfigureCursorBlink  
    LCD, 102  
ConfigureCursorOnOff  
    LCD, 103  
ConfigureDisplayOnOff  
    LCD, 104  
ConfigureEntryModeDir  
    LCD, 104  
ConfigureEntryModeShift  
    LCD, 105  
ConfigureFontType  
    LCD, 106  
ConfigureI2CAddr  
    LCD, 106  
ConfigureLineNumber  
    LCD, 107  
configureTimer1  
    timer, 178  
configureTimer3  
    timer, 179  
conversionTimerInterrupt  
    Bmp180, 16  
CpuLoad, 28  
    avg\_load, 31  
    ComputeCPUload, 29  
    CpuLoad, 28  
    current\_load, 31  
    getAverageCPUload, 29  
    getCurrentCPUload, 30  
    getMaxCPUload, 30  
    last\_sum\_value, 31  
    max\_load, 32  
    sample\_cnt, 32  
    sample\_idx, 32  
    sample\_mem, 32  
CpuLoad.cpp, 209  
    p\_global\_BSW\_cpupload, 210  
CpuLoad.h, 210  
    NB\_OF\_SAMPLES, 211  
    p\_global\_BSW\_cpupload, 211  
curElement\_ptr  
    LinkedList, 122  
current\_load  
    CpuLoad, 31  
cursor\_en  
    T\_LCD\_conf\_struct, 168  
cursorBlink\_en  
    T\_LCD\_conf\_struct, 168  
DATA\_ACK  
    I2C.h, 243  
DDRA\_PTR  
    dio\_reg\_atm2560.h, 229  
DDR\_B\_PTR  
    dio\_reg\_atm2560.h, 230  
DDRC\_PTR  
    dio\_reg\_atm2560.h, 230  
DDRD\_PTR  
    dio\_reg\_atm2560.h, 230  
DEBUG\_ACTIVE\_PORT  
    main.cpp, 265  
DECODE\_PIN  
    dio.h, 225  
DECODE\_PORT  
    dio.h, 225  
DHT22\_PORT  
    HumSensor.cpp, 240  
    TempSensor.cpp, 285  
DISPLAY\_LINE\_SHIFT\_PERIOD\_MS  
    DisplayInterface.h, 234  
DISPLAY\_LINE\_SHIFT\_TEMPO\_TIME  
    DisplayInterface.h, 234  
DISPLAY\_MGT\_FIRST\_LINE\_SENSORS  
    DisplayManagement.h, 238  
DISPLAY\_MGT\_I2C\_BITRATE  
    DisplayManagement.h, 238  
DISPLAY\_MGT\_LCD\_I2C\_ADDR  
    DisplayManagement.h, 238  
DISPLAY\_MGT\_PERIOD\_TASK\_SENSOR  
    DisplayManagement.h, 238  
DISPLAY\_MGT\_PERIOD\_WELCOME\_MSG\_REMOTE\_VAL  
    DisplayManagement.h, 239  
data\_name\_str  
    T\_SensorManagement\_Sensor\_Config, 171  
data\_ptr  
    LinkedList::T\_LL\_element, 170  
ddram\_addr  
    LCD, 114  
debug\_ift\_ptr  
    DebugManagement, 51  
debug\_mgt\_main\_menu\_state\_t  
    DebugManagement.h, 219  
debug\_mgt\_state\_struct\_t, 33  
    main\_state, 33  
    wdg\_state, 33  
debug\_mgt\_wdg\_state\_t  
    DebugManagement.h, 219  
debug\_state  
    DebugManagement, 51  
DebugInterface, 34  
    ClearScreen, 35

DebugInterface, 35  
 nextLine, 35  
 read, 36  
 sendBool, 36  
 sendChar, 37  
 sendInteger, 38  
 sendString, 39, 40  
 usart\_drv\_ptr, 40  
 DebugInterface.cpp, 212  
 p\_global\_ASW\_DebugInterface, 212  
 DebugInterface.h, 213  
 p\_global\_ASW\_DebugInterface, 214  
 USART\_BAUDRATE, 213  
 DebugManagement, 41  
 debug\_ift\_ptr, 51  
 debug\_state, 51  
 DebugManagement, 42  
 DebugModeManagement, 43  
 DisplayData, 44  
 DisplayPeriodicData\_task, 45  
 exitDebugMenu, 46  
 getIfPtr, 47  
 getInfoStringPtr, 47  
 getMenuStringPtr, 47  
 info\_string\_ptr, 51  
 isInfoStringDisplayed, 52  
 MainMenuManagement, 48  
 menu\_string\_ptr, 52  
 sensorMgt\_ptr, 52  
 setInfoStringPtr, 49  
 systemReset, 49  
 WatchdogMenuManagement, 50  
 DebugManagement.cpp, 214  
 p\_global\_ASW\_DebugManagement, 215  
 str\_debug\_info\_message\_wdg\_disabled, 215  
 str\_debug\_info\_message\_wdg\_enabled, 216  
 str\_debug\_info\_message\_wdg\_tmo\_updated, 216  
 str\_debug\_info\_message\_wdg\_tmo\_value, 216  
 str\_debug\_info\_message\_wrong\_menu\_selection,  
     216  
 str\_debug\_main\_menu, 216  
 str\_debug\_wdg\_menu, 217  
 str\_debug\_wdg\_timeout\_update\_selection, 217  
 DebugManagement.h, 218  
 debug\_mgt\_main\_menu\_state\_t, 219  
 debug\_mgt\_wdg\_state\_t, 219  
 p\_global\_ASW\_DebugManagement, 220  
 PERIOD\_MS\_TASK\_DISPLAY\_CPU\_LOAD, 219  
 PERIOD\_MS\_TASK\_DISPLAY\_DEBUG\_DATA,  
     219  
 DebugModeManagement  
     DebugManagement, 43  
 dht22, 53  
 dht22, 54  
 dht22\_port, 59  
 dio\_ptr, 59  
 getHumidity, 54  
 getTemperature, 56  
 initializeCommunication, 57  
 mem\_humidity, 59  
 mem\_temperature, 59  
 mem\_validity, 59  
 pit\_last\_read, 59  
 read, 58  
 dht22.cpp, 220  
 MAX\_WAIT\_TIME\_US, 221  
 p\_global\_BSW\_dht22, 221  
 dht22.h, 222  
 p\_global\_BSW\_dht22, 222  
 dht22\_port  
     dht22, 59  
 dio, 60  
 dio, 61  
 dio\_changePortPinCnf, 61  
 dio\_getPort, 62  
 dio\_getPort\_fast, 63  
 dio\_invertPort, 63  
 dio\_memorizePINaddress, 64  
 dio\_setPort, 65  
 getDDRxAddress, 66  
 getPINxAddress, 67  
 getPORTxAddress, 67  
 PINx\_addr\_mem, 68  
 PINx\_idx\_mem, 69  
 ports\_init, 68  
 dio.cpp, 223  
 p\_global\_BSW\_dio, 223  
 dio.h, 224  
 DECODE\_PIN, 225  
 DECODE\_PORT, 225  
 ENCODE\_PORT, 225  
 p\_global\_BSW\_dio, 226  
 PORT\_CNF\_IN, 225  
 PORT\_CNF\_OUT, 226  
 dio\_changePortPinCnf  
     dio, 61  
 dio\_getPort  
     dio, 62  
 dio\_getPort\_fast  
     dio, 63  
 dio\_invertPort  
     dio, 63  
 dio\_memorizePINaddress  
     dio, 64  
 dio\_port\_cnf.h, 226  
     PORT\_A, 227  
     PORT\_B, 227  
     PORT\_C, 227  
     PORT\_D, 228  
     PORTB\_CNF\_DDRB, 228  
     PORTB\_CNF\_PORTB, 228  
 dio\_ptr  
     dht22, 59  
 dio\_reg\_atm2560.h, 229  
     DDRA\_PTR, 229  
     DDRB\_PTR, 230

DDRC\_PTR, 230  
DDRD\_PTR, 230  
PINA\_PTR, 230  
PINB\_PTR, 230  
PINC\_PTR, 231  
PIND\_PTR, 231  
PORTA\_PTR, 231  
PORTB\_PTR, 231  
PORTC\_PTR, 231  
PORTD\_PTR, 232  
dio\_setPort  
    dio, 65  
disable  
    Watchdog, 190  
display\_data  
    DisplayInterface, 81  
display\_en  
    T\_LCD\_conf\_struct, 168  
display\_str  
    T\_display\_data, 165  
DisplayData  
    DebugManagement, 44  
DisplayFullLine  
    DisplayInterface, 74, 75  
DisplayInterface, 69  
    ClearFullScreen, 71  
    ClearLine, 72  
    ClearStringInDataStruct, 73  
    display\_data, 81  
    DisplayFullLine, 74, 75  
    DisplayInterface, 71  
    dummy, 81  
    FindFirstCharAddr, 75  
    getDisplayDataPtr, 76  
    IsLineEmpty, 77  
    isShiftInProgress, 81  
    p\_lcd, 81  
    RefreshLine, 77  
    setLineAlignment, 78  
    setLineAlignmentAndRefresh, 79  
    shiftLine\_task, 79  
    updateLineAndRefresh, 80  
DisplayInterface.cpp, 232  
    p\_global\_ASW\_DisplayInterface, 233  
DisplayInterface.h, 233  
    DISPLAY\_LINE\_SHIFT\_PERIOD\_MS, 234  
    DISPLAY\_LINE\_SHIFT\_TEMPO\_TIME, 234  
    p\_global\_ASW\_DisplayInterface, 235  
    T\_DisplayInterface\_LineAlignment, 234  
    T\_DisplayInterface\_LineDisplayMode, 235  
DisplayManagement, 82  
    DisplayManagement, 83  
    DisplaySensorData\_Task, 84  
    GetIftPointer, 84  
    GetSensorMgtPtr, 85  
    p\_SensorMgt, 86  
    p\_display\_ift, 86  
    RemoveWelcomeMessage\_Task, 85  
    DisplayManagement.cpp, 236  
        noSensorsDisplayString, 236  
        p\_global\_ASW\_DisplayManagement, 236  
        welcomeMessageString, 237  
    DisplayManagement.h, 237  
        DISPLAY\_MGT\_FIRST\_LINE\_SENSORS, 238  
        DISPLAY\_MGT\_I2C\_BITRATE, 238  
        DISPLAY\_MGT\_LCD\_I2C\_ADDR, 238  
        DISPLAY\_MGT\_PERIOD\_TASK\_SENSOR, 238  
        DISPLAY\_MGT\_PERIOD\_WELCOME\_MSG\_R←  
            EMOVAL, 239  
        LCD\_init\_cnf, 239  
        p\_global\_ASW\_DisplayManagement, 239  
    DisplayPeriodicData\_task  
        DebugManagement, 45  
    DisplaySensorData\_Task  
        DisplayManagement, 84  
    dummy  
        DisplayInterface, 81  
    EN\_PIN  
        LCD.h, 253  
    ENCODE\_PORT  
        dio.h, 225  
    enable  
        Watchdog, 191  
    entryModeDir  
        T\_LCD\_conf\_struct, 168  
    entryModeShift  
        T\_LCD\_conf\_struct, 168  
    exitDebugMenu  
        DebugManagement, 46  
    FindElement  
        LinkedList, 118  
    FindFirstCharAddr  
        DisplayInterface, 75  
    firstElement  
        LinkedList, 122  
    fontType\_cnf  
        T\_LCD\_conf\_struct, 169  
    getAverageCPUload  
        CpuLoad, 29  
    getCurrentElement  
        LinkedList, 119  
    getCurrentCPUload  
        CpuLoad, 30  
    GetDDRAMAddress  
        LCD, 108  
    getDDRxAddress  
        dio, 66  
    getDisplayDataPtr  
        DisplayInterface, 76  
    getFullStringFormattedValue  
        SensorManagement, 145  
    getHumidity  
        dht22, 54  
    GetIftPointer

DisplayManagement, 84  
 getIftPtr  
     DebugManagement, 47  
 getInfoStringPtr  
     DebugManagement, 47  
 GetLineNumberCnf  
     LCD, 108  
 getMaxCPUload  
     CpuLoad, 30  
 getMenuStringPtr  
     DebugManagement, 47  
 getMonitoringTaskPeriod  
     Bmp180, 17  
 getPINxAddress  
     dio, 67  
 getPORTxAddress  
     dio, 67  
 getPitNumber  
     scheduler, 130  
 getPressureValue  
     Bmp180, 17  
 getRawDataPtr  
     Sensor, 138  
 getSensorCount  
     SensorManagement, 146  
 GetSensorMgtPtr  
     DisplayManagement, 85  
 getSensorObjectPtr  
     SensorManagement, 146  
 getSize  
     String, 157  
 getStatus  
     Bmp180, 18  
 getString  
     String, 157  
 getTMOValue  
     Watchdog, 191  
 getTaskCount  
     scheduler, 131  
 getTaskPeriod  
     Sensor, 138  
 getTemperature  
     dht22, 56  
 getTemperatureValue  
     Bmp180, 18  
 getTimer1Value  
     timer, 180  
 getValidity  
     Sensor, 139  
 getValue  
     Sensor, 139  
 getValueDecimal  
     Sensor, 139  
 getValueInteger  
     Sensor, 140  
 HumSensor, 87  
     HumSensor, 88  
 readHumSensor\_task, 89  
 updateTaskPeriod, 90  
 HumSensor.cpp, 240  
     DHT22\_PORT, 240  
 HumSensor.h, 241  
 I2C.cpp, 241  
     p\_global\_BSW\_i2c, 242  
 I2C.h, 242  
     DATA\_ACK, 243  
     p\_global\_BSW\_i2c, 244  
     REPEATED\_START, 243  
     SLAR\_ACK, 243  
     SLAW\_ACK, 244  
     START, 244  
 I2C, 91  
     bitrate, 96  
     I2C, 92  
     initializeBus, 92  
     read, 93  
     setBitRate, 94  
     write, 94  
     writeByte, 95  
 i2c\_addr  
     T\_LCD\_conf\_struct, 169  
 i2c\_bitrate  
     T\_LCD\_conf\_struct, 169  
 i2c\_drv\_ptr  
     Bmp180, 26  
     LCD, 114  
 ISR  
     int.cpp, 245–247  
 info\_string\_ptr  
     DebugManagement, 51  
 initializeBus  
     I2C, 92  
 initializeCommunication  
     dht22, 57  
 InitializeScreen  
     LCD, 108  
 int.cpp, 245  
     ISR, 245–247  
 isActive  
     Watchdog, 195  
 isDebugEnabled  
     T\_ASW\_init\_cnf, 159  
 isDebugModeActivated  
     main.cpp, 266  
     main.h, 267  
 isDisplayActivated  
     T\_ASW\_init\_cnf, 159  
 isEmpty  
     T\_display\_data, 165  
 isEnabled  
     Watchdog, 192  
 isInfoStringDisplayed  
     DebugManagement, 52  
 isLEDActivated  
     T\_ASW\_init\_cnf, 159  
 IsLLEmpty

LinkedList, 119  
    IsLineEmpty  
        DisplayInterface, 77  
    isPressConvActivated  
        Bmp180, 26  
    isPressConversionActivated  
        Bmp180, 19  
    isSensorMgtActivated  
        T\_ASW\_init\_cnf, 160  
    isShiftInProgress  
        DisplayInterface, 81  
    isTempConvActivated  
        Bmp180, 26  
    isTempConversionActivated  
        Bmp180, 19  
  
keepAliveLed, 96  
    blinkLed\_task, 97  
    keepAliveLed, 97  
keepAliveLed.cpp, 248  
    p\_global\_ASW\_keepAliveLed, 248  
keepAliveLed.h, 249  
    LED\_PORT, 249  
    p\_global\_ASW\_keepAliveLed, 250  
    PERIOD\_MS\_TASK\_LED, 250  
  
LCD.cpp, 250  
    p\_global\_BSW\_lcd, 251  
LCD.h, 251  
    BACKLIGHT\_PIN, 253  
    EN\_PIN, 253  
    LCD\_CNF\_BACKLIGHT\_OFF, 253  
    LCD\_CNF\_BACKLIGHT\_ON, 253  
    LCD\_CNF\_CURSOR\_BLINK\_OFF, 254  
    LCD\_CNF\_CURSOR\_BLINK\_ON, 254  
    LCD\_CNF\_CURSOR\_OFF, 254  
    LCD\_CNF\_CURSOR\_ON, 254  
    LCD\_CNF\_DISPLAY\_OFF, 254  
    LCD\_CNF\_DISPLAY\_ON, 255  
    LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_LEFT,  
        255  
    LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_RIGHT,  
        255  
    LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT\_↔  
        OFF, 255  
    LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT\_↔  
        ON, 255  
    LCD\_CNF\_FONT\_5\_11, 256  
    LCD\_CNF\_FONT\_5\_8, 256  
    LCD\_CNF\_ONE\_LINE, 256  
    LCD\_CNF\_SHIFT\_ID, 256  
    LCD\_CNF\_SHIFT\_SH, 256  
    LCD\_CNF\_TWO\_LINE, 257  
    LCD\_DISPLAY\_CTRL\_FIELD\_B, 257  
    LCD\_DISPLAY\_CTRL\_FIELD\_C, 257  
    LCD\_DISPLAY\_CTRL\_FIELD\_D, 257  
    LCD\_FCT\_SET\_FIELD\_DL, 257  
    LCD\_FCT\_SET\_FIELD\_F, 258  
    LCD\_FCT\_SET\_FIELD\_N, 258  
  
    LCD\_INST\_CLR\_DISPLAY\_BIT, 258  
    LCD\_INST\_DISPLAY\_CTRL, 258  
    LCD\_INST\_ENTRY\_MODE\_SET, 258  
    LCD\_INST\_FUNCTION\_SET, 259  
    LCD\_INST\_SET\_DDRAM\_ADDR, 259  
    LCD\_RAM\_1\_LINE\_MAX, 259  
    LCD\_RAM\_1\_LINE\_MIN, 259  
    LCD\_RAM\_2\_LINES\_MAX\_1, 259  
    LCD\_RAM\_2\_LINES\_MAX\_2, 260  
    LCD\_RAM\_2\_LINES\_MIN\_1, 260  
    LCD\_RAM\_2\_LINES\_MIN\_2, 260  
    LCD\_SIZE\_NB\_CHAR\_PER\_LINE, 260  
    LCD\_SIZE\_NB\_LINES, 260  
    LCD\_WAIT\_CLR\_RETURN, 261  
    LCD\_WAIT\_OTHER\_MODES, 261  
    p\_global\_BSW\_lcd, 262  
    RS\_PIN, 261  
    RW\_PIN, 261  
    T\_LCD\_command, 261  
    T\_LCD\_config\_mode, 262  
    T\_LCD\_ram\_area, 262  
LCD\_CNF\_BACKLIGHT\_OFF  
    LCD.h, 253  
LCD\_CNF\_BACKLIGHT\_ON  
    LCD.h, 253  
LCD\_CNF\_CURSOR\_BLINK\_OFF  
    LCD.h, 254  
LCD\_CNF\_CURSOR\_BLINK\_ON  
    LCD.h, 254  
LCD\_CNF\_CURSOR\_OFF  
    LCD.h, 254  
LCD\_CNF\_CURSOR\_ON  
    LCD.h, 254  
LCD\_CNF\_DISPLAY\_OFF  
    LCD.h, 254  
LCD\_CNF\_DISPLAY\_ON  
    LCD.h, 255  
LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_LEFT  
    LCD.h, 255  
LCD\_CNF\_ENTRY\_MODE\_DIRECTION\_RIGHT  
    LCD.h, 255  
LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT\_OFF  
    LCD.h, 255  
LCD\_CNF\_ENTRY\_MODE\_DISPLAY\_SHIFT\_ON  
    LCD.h, 255  
LCD\_CNF\_FONT\_5\_11  
    LCD.h, 256  
LCD\_CNF\_FONT\_5\_8  
    LCD.h, 256  
LCD\_CNF\_ONE\_LINE  
    LCD.h, 256  
LCD\_CNF\_SHIFT\_ID  
    LCD.h, 256  
LCD\_CNF\_SHIFT\_SH  
    LCD.h, 256  
LCD\_CNF\_TWO\_LINE  
    LCD.h, 257  
LCD\_DISPLAY\_CTRL\_FIELD\_B

LCD.h, 257  
 LCD\_DISPLAY\_CTRL\_FIELD\_C  
   LCD.h, 257  
 LCD\_DISPLAY\_CTRL\_FIELD\_D  
   LCD.h, 257  
 LCD\_FCT\_SET\_FIELD\_DL  
   LCD.h, 257  
 LCD\_FCT\_SET\_FIELD\_F  
   LCD.h, 258  
 LCD\_FCT\_SET\_FIELD\_N  
   LCD.h, 258  
 LCD\_INST\_CLR\_DISPLAY\_BIT  
   LCD.h, 258  
 LCD\_INST\_DISPLAY\_CTRL  
   LCD.h, 258  
 LCD\_INST\_ENTRY\_MODE\_SET  
   LCD.h, 258  
 LCD\_INST\_FUNCTION\_SET  
   LCD.h, 259  
 LCD\_INST\_SET\_DDRAM\_ADDR  
   LCD.h, 259  
 LCD\_RAM\_1\_LINE\_MAX  
   LCD.h, 259  
 LCD\_RAM\_1\_LINE\_MIN  
   LCD.h, 259  
 LCD\_RAM\_2\_LINES\_MAX\_1  
   LCD.h, 259  
 LCD\_RAM\_2\_LINES\_MAX\_2  
   LCD.h, 260  
 LCD\_RAM\_2\_LINES\_MIN\_1  
   LCD.h, 260  
 LCD\_RAM\_2\_LINES\_MIN\_2  
   LCD.h, 260  
 LCD\_SIZE\_NB\_CHAR\_PER\_LINE  
   LCD.h, 260  
 LCD\_SIZE\_NB\_LINES  
   LCD.h, 260  
 LCD\_WAIT\_CLR\_RETURN  
   LCD.h, 261  
 LCD\_WAIT\_OTHER\_MODES  
   LCD.h, 261  
 LCD\_init\_cnf  
   DisplayManagement.h, 239  
 LCD, 98  
   backlight\_enable, 113  
   cnfCursorBlink, 113  
   cnfCursorOnOff, 113  
   cnfDisplayOnOff, 113  
   cnfEntryModeDir, 113  
   cnfEntryModeShift, 113  
   cnfFontType, 114  
   cnfI2C\_addr, 114  
   cnfLineNumber, 114  
   command, 101  
   ConfigureBacklight, 102  
   ConfigureCursorBlink, 102  
   ConfigureCursorOnOff, 103  
   ConfigureDisplayOnOff, 104  
     ConfigureEntryModeDir, 104  
     ConfigureEntryModeShift, 105  
     ConfigureFontType, 106  
     ConfigureI2CAddr, 106  
     ConfigureLineNumber, 107  
     ddram\_addr, 114  
     GetDDRAMAddress, 108  
     GetLineNumberCnf, 108  
     i2c\_drv\_ptr, 114  
     InitializeScreen, 108  
     LCD, 100  
     SetDDRAMAddress, 109  
     write, 110  
     write4bits, 111  
     WriteInRam, 112  
 LED\_PORT  
   keepAliveLed.h, 249  
 LL\_ElementCompare  
   scheduler, 132  
 last\_sum\_value  
   CpuLoad, 31  
 launchPeriodicTasks  
   scheduler, 131  
 lineNumber\_cnf  
   T\_LCD\_conf\_struct, 169  
 LinkedList, 115  
   ~LinkedList, 117  
   AttachNewElement, 118  
   curElement\_ptr, 122  
   FindElement, 118  
   firstElement, 122  
   getCurrentElement, 119  
   IsLLEmpty, 119  
   LinkedList, 117  
   MoveToNextElement, 120  
   RemoveElement, 120  
   ResetElementPtr, 121  
   T\_LL\_element, 116  
 LinkedList.cpp, 263  
 LinkedList.h, 263  
   CompareFctPtr\_t, 264  
 LinkedList::T\_LL\_element, 170  
   data\_ptr, 170  
   nextElement, 170  
 MAX\_WAIT\_TIME\_US  
   dht22.cpp, 221  
 main  
   main.cpp, 265  
 main.cpp, 264  
   ASW\_init\_cnf, 266  
   DEBUG\_ACTIVE\_PORT, 265  
   isDebugModeActivated, 266  
   main, 265  
 main.h, 267  
   ASW\_init\_cnf, 267  
   isDebugModeActivated, 267  
 main\_state  
   debug\_mgt\_state\_struct\_t, 33

MainMenuManagement  
    DebugManagement, 48

max\_load  
    CpuLoad, 32

MB  
    Bmp180::T\_BMP180\_calib\_data, 162

MC  
    Bmp180::T\_BMP180\_calib\_data, 162

MD  
    Bmp180::T\_BMP180\_calib\_data, 162

mem\_humidity  
    dht22, 59

mem\_temperature  
    dht22, 59

mem\_validity  
    dht22, 59

menu\_string\_ptr  
    DebugManagement, 52

mode  
    T\_display\_data, 165

MoveToNextElement  
    LinkedList, 120

NB\_OF\_SAMPLES  
    CpuLoad.h, 211

nb\_sensors  
    SensorManagement, 148

nextElement  
    LinkedList::T\_LL\_element, 170

nextLine  
    DebugInterface, 35

noSensorsDisplayString  
    DisplayManagement.cpp, 236

operator delete  
    operators.cpp, 269  
    operators.h, 270

operator new  
    operators.cpp, 269  
    operators.h, 271

operators.cpp, 268  
    operator delete, 269  
    operator new, 269

operators.h, 269  
    operator delete, 270  
    operator new, 271

p\_SensorMgt  
    DisplayManagement, 86

p\_display\_ift  
    DisplayManagement, 86

p\_global\_ASW\_DebugInterface  
    DebugInterface.cpp, 212  
    DebugInterface.h, 214

p\_global\_ASW\_DebugManagement  
    DebugManagement.cpp, 215  
    DebugManagement.h, 220

p\_global\_ASW\_DisplayInterface  
    DisplayInterface.cpp, 233

    DisplayInterface.h, 235

p\_global\_ASW\_DisplayManagement  
    DisplayManagement.cpp, 236  
    DisplayManagement.h, 239

p\_global\_ASW\_SensorManagement  
    SensorManagement.cpp, 281  
    SensorManagement.h, 283

p\_global\_ASW\_keepAliveLed  
    keepAliveLed.cpp, 248  
    keepAliveLed.h, 250

p\_global\_BSW\_bmp180  
    Bmp180.cpp, 201  
    Bmp180.h, 206

p\_global\_BSW\_cupload  
    CpuLoad.cpp, 210  
    CpuLoad.h, 211

p\_global\_BSW\_dht22  
    dht22.cpp, 221  
    dht22.h, 222

p\_global\_BSW\_dio  
    dio.cpp, 223  
    dio.h, 226

p\_global\_BSW\_i2c  
    I2C.cpp, 242  
    I2C.h, 244

p\_global\_BSW\_lcd  
    LCD.cpp, 251  
    LCD.h, 262

p\_global\_BSW\_timer  
    timer.cpp, 287  
    timer.h, 288

p\_global\_BSW\_usart  
    usart.cpp, 289  
    usart.h, 290

p\_global\_BSW\_wdg  
    Watchdog.cpp, 291  
    Watchdog.h, 294

p\_global\_scheduler  
    scheduler.cpp, 273  
    scheduler.h, 275

p\_lcd  
    DisplayInterface, 81

PERIOD\_MS\_TASK\_DISPLAY\_CPU\_LOAD  
    DebugManagement.h, 219

PERIOD\_MS\_TASK\_DISPLAY\_DEBUG\_DATA  
    DebugManagement.h, 219

PERIOD\_MS\_TASK\_LED  
    keepAliveLed.h, 250

PINA\_PTR  
    dio\_reg\_atm2560.h, 230

PINB\_PTR  
    dio\_reg\_atm2560.h, 230

PINC\_PTR  
    dio\_reg\_atm2560.h, 231

PIND\_PTR  
    dio\_reg\_atm2560.h, 231

PINx\_addr\_mem  
    dio, 68

PINx\_idx\_mem  
     dio, 69  
 PORT\_CNF\_IN  
     dio.h, 225  
 PORT\_CNF\_OUT  
     dio.h, 226  
 PORT\_A  
     dio\_port\_cnf.h, 227  
 PORT\_B  
     dio\_port\_cnf.h, 227  
 PORT\_C  
     dio\_port\_cnf.h, 227  
 PORT\_D  
     dio\_port\_cnf.h, 228  
 PORTA\_PTR  
     dio\_reg\_atm2560.h, 231  
 PORTB\_CNF\_DDRB  
     dio\_port\_cnf.h, 228  
 PORTB\_CNF\_PORTB  
     dio\_port\_cnf.h, 228  
 PORTB\_PTR  
     dio\_reg\_atm2560.h, 231  
 PORTC\_PTR  
     dio\_reg\_atm2560.h, 231  
 PORTD\_PTR  
     dio\_reg\_atm2560.h, 232  
 PRESCALER\_PERIODIC\_TIMER  
     scheduler.h, 274  
 period  
     scheduler::Task\_t, 172  
     T\_SensorManagement\_Sensor\_Config, 171  
 pit\_last\_read  
     dht22, 59  
 pit\_number  
     scheduler, 135  
 ports\_init  
     dio, 68  
 prescaler1  
     timer, 182  
 prescaler3  
     timer, 182  
 PressSensor, 122  
     PressSensor, 123, 124  
     readPressSensor\_task, 125  
     updateTaskPeriod, 126  
 PressSensor.cpp, 271  
 PressSensor.h, 272  
 pressure\_value  
     Bmp180, 27  
 PressureMonitoring  
     Bmp180, 20  
 REPEATED\_START  
     I2C.h, 243  
 RS\_PIN  
     LCD.h, 261  
 RW\_PIN  
     LCD.h, 261  
 raw\_data  
     Sensor, 143  
 read  
     DebugInterface, 36  
     dht22, 58  
     I2C, 93  
 readCalibData  
     Bmp180, 20  
 readChipID  
     Bmp180, 21  
 readHumSensor\_task  
     HumSensor, 89  
 readPressSensor\_task  
     PressSensor, 125  
 readSensor\_task  
     Sensor, 140  
 readTempSensor\_task  
     TempSensor, 176  
 ready  
     Bmp180::T\_BMP180\_measurement\_data, 163  
 RefreshLine  
     DisplayInterface, 77  
 RemoveElement  
     LinkedList, 120  
 removePeriodicTask  
     scheduler, 132  
 RemoveWelcomeMessage\_Task  
     DisplayManagement, 85  
 reset  
     Watchdog, 192  
 ResetElementPtr  
     LinkedList, 121  
 SENSOR\_MGT\_CNF\_DEFAULT\_PERIOD  
     sensor\_configuration.cpp, 278  
 SENSOR\_MGT\_CNF\_DEFAULT\_TMO  
     sensor\_configuration.cpp, 279  
 SLAR\_ACK  
     I2C.h, 243  
 SLAW\_ACK  
     I2C.h, 244  
 START  
     I2C.h, 244  
 SW\_PERIOD\_MS  
     scheduler.h, 275  
 sample\_cnt  
     CpuLoad, 32  
 sample\_idx  
     CpuLoad, 32  
 sample\_mem  
     CpuLoad, 32  
 scheduler, 127  
     addPeriodicTask, 129  
     getPitNumber, 130  
     getTaskCount, 131  
     LLElementCompare, 132  
     launchPeriodicTasks, 131  
     pit\_number, 135  
     removePeriodicTask, 132  
     scheduler, 128

startScheduling, 133  
task\_count, 135  
Task\_t, 128  
TasksLL\_ptr, 135  
updateTaskPeriod, 134  
scheduler.cpp, 273  
    p\_global\_scheduler, 273  
scheduler.h, 274  
    p\_global\_scheduler, 275  
PRESCALER\_PERIODIC\_TIMER, 274  
SW\_PERIOD\_MS, 275  
TIMER\_CTC\_VALUE, 275  
TaskPtr\_t, 275  
scheduler::Task\_t, 172  
    period, 172  
    TaskPtr, 172  
sendBool  
    DebugInterface, 36  
sendChar  
    DebugInterface, 37  
sendInteger  
    DebugInterface, 38  
sendString  
    DebugInterface, 39, 40  
Sensor, 136  
    getRawDataPtr, 138  
    getTaskPeriod, 138  
    getValidity, 139  
    getValue, 139  
    getValueDecimal, 139  
    getValueInteger, 140  
    raw\_data, 143  
    readSensor\_task, 140  
    Sensor, 137  
    setLastValidity, 140  
    setValidityTMO, 141  
    task\_period, 143  
    updateTaskPeriod, 141  
    updateValidData, 142  
    valid\_pit, 143  
    valid\_value, 143  
    validity, 143  
    validity\_last\_read, 143  
    validity\_tmo, 144  
Sensor.cpp, 276  
    TASK\_PERIOD\_DEFAULT, 276  
    VALIDITY\_TIMEOUT\_MS\_DEFAULT, 277  
Sensor.h, 277  
sensor\_configuration.cpp, 278  
    SENSOR\_MGT\_CNF\_DEFAULT\_PERIOD, 278  
    SENSOR\_MGT\_CNF\_DEFAULT\_TMO, 279  
    SensorManagement\_Sensor\_Config\_list, 279  
sensor\_configuration.h, 280  
    SensorManagement\_Sensor\_Config\_list, 280  
sensor\_ptr\_table  
    SensorManagement, 149  
sensor\_type  
    T\_SensorManagement\_Sensor\_Config, 171  
SensorManagement, 144  
    getFullStringFormattedValue, 145  
    getSensorCount, 146  
    getSensorObjectPtr, 146  
    nb\_sensors, 148  
    sensor\_ptr\_table, 149  
    SensorManagement, 145  
    updateTaskPeriod, 148  
SensorManagement.cpp, 281  
    p\_global\_ASW\_SensorManagement, 281  
SensorManagement.h, 282  
    p\_global\_ASW\_SensorManagement, 283  
    T\_SensorManagement\_Sensor\_Type, 282  
SensorManagement\_Sensor\_Config\_list  
    sensor\_configuration.cpp, 279  
    sensor\_configuration.h, 280  
sensorMgt\_ptr  
    DebugManagement, 52  
setBaudRate  
    uart, 184  
setBitRate  
    I2C, 94  
SetDDRAMAddress  
    LCD, 109  
setInfoStringPtr  
    DebugManagement, 49  
setLastValidity  
    Sensor, 140  
setLineAlignment  
    DisplayInterface, 78  
setLineAlignmentAndRefresh  
    DisplayInterface, 79  
setValidityTMO  
    Sensor, 141  
shift\_data  
    T\_display\_data, 165  
shiftLine\_task  
    DisplayInterface, 79  
size  
    String, 158  
startNewPressureConversion  
    Bmp180, 22  
startNewTemperatureConversion  
    Bmp180, 23  
startScheduling  
    scheduler, 133  
startTimer1  
    timer, 180  
startTimer3  
    timer, 181  
status  
    Bmp180, 27  
StopPressureConversion  
    Bmp180, 24  
StopTemperatureConversion  
    Bmp180, 24  
stopTimer1  
    timer, 181

stopTimer3  
     timer, 181

str\_cur\_ptr  
     T\_Display\_shift\_data, 166

str\_debug\_info\_message\_wdg\_disabled  
     DebugManagement.cpp, 215

str\_debug\_info\_message\_wdg\_enabled  
     DebugManagement.cpp, 216

str\_debug\_info\_message\_wdg\_tmo\_updated  
     DebugManagement.cpp, 216

str\_debug\_info\_message\_wdg\_tmo\_value  
     DebugManagement.cpp, 216

str\_debug\_info\_message\_wrong\_menu\_selection  
     DebugManagement.cpp, 216

str\_debug\_main\_menu  
     DebugManagement.cpp, 216

str\_debug\_wdg\_menu  
     DebugManagement.cpp, 217

str\_debug\_wdg\_timeout\_update\_selection  
     DebugManagement.cpp, 217

str\_ptr  
     T\_Display\_shift\_data, 166

String, 149  
     ~String, 151  
     appendBool, 152  
     appendChar, 152  
     appendInteger, 153  
     appendSpace, 154  
     appendString, 155  
     Clear, 156  
     ComputeStringSize, 156  
     getSize, 157  
     getString, 157  
     size, 158  
     String, 150, 151  
     string, 158

string  
     String, 158

String.cpp, 283

String.h, 284

SwitchWdg  
     Watchdog, 193

SystemReset  
     Watchdog, 194

systemReset  
     DebugManagement, 49

T\_ASW\_init\_cnf, 159  
     isDebugEnabled, 159  
     isDisplayEnabled, 159  
     isLEDEnabled, 159  
     isSensorMgtEnabled, 160

T\_BMP180\_status  
     Bmp180.h, 206

T\_Display\_shift\_data, 166  
     str\_cur\_ptr, 166  
     str\_ptr, 166  
     temporization, 167

T\_DisplayInterface\_LineAlignment

    DisplayInterface.h, 234

    T\_DisplayInterface\_LineDisplayMode  
         DisplayInterface.h, 235

    T\_LCD\_Command  
         LCD.h, 261

    T\_LCD\_conf\_struct, 167  
         backlight\_en, 168  
         cursor\_en, 168  
         cursorBlink\_en, 168  
         display\_en, 168  
         entryModeDir, 168  
         entryModeShift, 168  
         fontType\_cnf, 169  
         i2c\_addr, 169  
         i2c\_bitrate, 169  
         lineNumber\_cnf, 169

    T\_LCD\_config\_mode  
         LCD.h, 262

    T\_LCD\_ram\_area  
         LCD.h, 262

    T\_LL\_element  
         LinkedList, 116

    T\_SensorManagement\_Sensor\_Config, 171  
         data\_name\_str, 171  
         period, 171  
         sensor\_type, 171  
         unit\_str, 171  
         validity\_tmo, 172

    T\_SensorManagement\_Sensor\_Type  
         SensorManagement.h, 282

    T\_display\_data, 164  
         alignment, 164  
         display\_str, 165  
         isEmpty, 165  
         mode, 165  
         shift\_data, 165

    TASK\_PERIOD\_DEFAULT  
         Sensor.cpp, 276

    TEMP\_SENSOR\_VALUE\_ACCEPTABLE\_DIFFERENCE  
         NCE  
         TempSensor.cpp, 285

    TIMER\_CTC\_VALUE  
         scheduler.h, 275

    task\_count  
         scheduler, 135

    task\_period  
         Bmp180, 27  
         Sensor, 143

    Task\_t  
         scheduler, 128

    TaskPtr  
         scheduler::Task\_t, 172

    TaskPtr\_t  
         scheduler.h, 275

    TasksLL\_ptr  
         scheduler, 135

    TempSensor, 173  
         readTempSensor\_task, 176

TempSensor, 174, 175  
updateTaskPeriod, 176  
TempSensor.cpp, 284  
    DHT22\_PORT, 285  
    TEMP\_SENSOR\_VALUE\_ACCEPTABLE\_DIFF←  
        ERENCE, 285  
TempSensor.h, 285  
temperature\_value  
    Bmp180, 27  
TemperatureMonitoring  
    Bmp180, 25  
temporization  
    T\_Display\_shift\_data, 167  
timeoutUpdate  
    Watchdog, 194  
timer, 177  
    configureTimer1, 178  
    configureTimer3, 179  
    getTimer1Value, 180  
    prescaler1, 182  
    prescaler3, 182  
    startTimer1, 180  
    startTimer3, 181  
    stopTimer1, 181  
    stopTimer3, 181  
    timer, 178  
timer.cpp, 286  
    p\_global\_BSW\_timer, 287  
timer.h, 287  
    p\_global\_BSW\_timer, 288  
tmo\_value  
    Watchdog, 195  
ts  
    Bmp180::T\_BMP180\_measurement\_data, 163  
USART\_BAUDRATE  
    DebugInterface.h, 213  
unit\_str  
    T\_SensorManagement\_Sensor\_Config, 171  
updateLineAndRefresh  
    DisplayInterface, 80  
updateTaskPeriod  
    HumSensor, 90  
    PressSensor, 126  
    scheduler, 134  
    Sensor, 141  
    SensorManagement, 148  
    TempSensor, 176  
updateValidData  
    Sensor, 142  
usart, 183  
    BaudRate, 188  
    setBaudRate, 184  
    usart, 183  
    usart\_init, 184  
    usart\_read, 185  
    usart\_sendByte, 185  
    usart\_sendString, 186  
    usart\_transmit, 187  
    uart.cpp, 288  
        p\_global\_BSW\_usart, 289  
usart.h, 289  
    p\_global\_BSW\_usart, 290  
usart\_drv\_ptr  
    DebugInterface, 40  
usart\_init  
    usart, 184  
usart\_read  
    usart, 185  
usart\_sendByte  
    usart, 185  
usart\_sendString  
    usart, 186  
usart\_transmit  
    usart, 187  
VALIDITY\_TIMEOUT\_MS\_DEFAULT  
    Sensor.cpp, 277  
valid\_pit  
    Sensor, 143  
valid\_value  
    Sensor, 143  
validity  
    Sensor, 143  
validity\_last\_read  
    Sensor, 143  
validity\_tmo  
    Sensor, 144  
    T\_SensorManagement\_Sensor\_Config, 172  
value  
    Bmp180::T\_BMP180\_measurement\_data, 163  
WDG\_TIMEOUT\_DEFAULT\_MS  
    Watchdog.cpp, 291  
WDG\_TMO\_120MS  
    Watchdog.h, 292  
WDG\_TMO\_15MS  
    Watchdog.h, 292  
WDG\_TMO\_1S  
    Watchdog.h, 293  
WDG\_TMO\_250MS  
    Watchdog.h, 293  
WDG\_TMO\_2S  
    Watchdog.h, 293  
WDG\_TMO\_30MS  
    Watchdog.h, 293  
WDG\_TMO\_4S  
    Watchdog.h, 293  
WDG\_TMO\_500MS  
    Watchdog.h, 294  
WDG\_TMO\_60MS  
    Watchdog.h, 294  
WDG\_TMO\_8S  
    Watchdog.h, 294  
Watchdog, 188  
    disable, 190  
    enable, 191  
    getTMOValue, 191

isActive, 195  
isEnabled, 192  
reset, 192  
SwitchWdg, 193  
SystemReset, 194  
timeoutUpdate, 194  
tmo\_value, 195  
Watchdog, 189  
Watchdog.cpp, 290  
  p\_global\_BSW\_wdg, 291  
  WDG\_TIMEOUT\_DEFAULT\_MS, 291  
Watchdog.h, 291  
  p\_global\_BSW\_wdg, 294  
  WDG\_TMO\_120MS, 292  
  WDG\_TMO\_15MS, 292  
  WDG\_TMO\_1S, 293  
  WDG\_TMO\_250MS, 293  
  WDG\_TMO\_2S, 293  
  WDG\_TMO\_30MS, 293  
  WDG\_TMO\_4S, 293  
  WDG\_TMO\_500MS, 294  
  WDG\_TMO\_60MS, 294  
  WDG\_TMO\_8S, 294  
WatchdogMenuManagement  
  DebugManagement, 50  
wdg\_state  
  debug\_mgt\_state\_struct\_t, 33  
welcomeMessageString  
  DisplayManagement.cpp, 237  
write  
  I2C, 94  
  LCD, 110  
write4bits  
  LCD, 111  
writeByte  
  I2C, 95  
WriteInRam  
  LCD, 112