

**Green House software**

1.0

Generated by Doxygen 1.9.1



---

<b>1 Hierarchical Index</b>	<b>1</b>
1.1 Class Hierarchy	1
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Class Documentation</b>	<b>9</b>
4.1 AirQualitySensor Class Reference	9
4.1.1 Detailed Description	12
4.1.2 Constructor & Destructor Documentation	12
4.1.2.1 AirQualitySensor()	12
4.1.2.2 ~AirQualitySensor()	12
4.1.3 Member Function Documentation	13
4.1.3.1 checkAllgood()	13
4.1.3.2 collectAndPrint()	14
4.1.3.3 collectData()	14
4.1.3.4 printData()	15
4.1.3.5 stringStatus()	16
4.1.4 Friends And Related Function Documentation	17
4.1.4.1 operator<<	17
4.2 AlarmSensors Class Reference	18
4.2.1 Detailed Description	19
4.2.2 Constructor & Destructor Documentation	19
4.2.2.1 AlarmSensors()	20
4.2.2.2 ~AlarmSensors()	20
4.2.3 Member Function Documentation	20
4.2.3.1 addSensor()	20
4.2.3.2 checkAllgood()	21
4.2.3.3 checkSensors()	22
4.2.3.4 deleteSensor()	22
4.2.3.5 displayAlarmStatus()	23
4.2.3.6 displayAllSensorsData()	24
4.2.3.7 loadSensorsData()	24
4.2.3.8 loadSensorsDataBin()	25
4.2.3.9 loadSensorsDataTxt()	25
4.2.3.10 saveSensorsData()	26
4.2.3.11 saveSensorsDataBin()	26
4.2.3.12 saveSensorsDataTxt()	26
4.2.3.13 sensorExists()	27
4.2.3.14 sensorsInitialized()	27

---

4.2.3.15 turnOffSystem()	28
4.2.3.16 turnOnOffSystem()	28
4.2.3.17 turnOnSystem()	29
4.2.4 Member Data Documentation	29
4.2.4.1 fileNameBin	29
4.2.4.2 fileNameTxt	29
4.2.4.3 sensors_	29
4.2.4.4 status_	30
4.3 Camera Class Reference	30
4.3.1 Detailed Description	32
4.3.2 Constructor & Destructor Documentation	32
4.3.2.1 Camera()	32
4.3.2.2 ~Camera()	32
4.3.3 Member Function Documentation	33
4.3.3.1 collectData()	33
4.3.3.2 getId()	33
4.3.3.3 getType()	34
4.3.3.4 isActive()	34
4.3.3.5 printCamera()	34
4.3.3.6 setId()	35
4.3.3.7 setType()	36
4.3.3.8 turnOff()	36
4.3.3.9 turnOn()	37
4.3.4 Member Data Documentation	37
4.3.4.1 active_	37
4.3.4.2 id_	37
4.3.4.3 type_	37
4.4 FileCloseError Class Reference	38
4.4.1 Detailed Description	39
4.4.2 Constructor & Destructor Documentation	39
4.4.2.1 FileCloseError()	39
4.5 FileCorruptError Class Reference	39
4.5.1 Detailed Description	40
4.5.2 Constructor & Destructor Documentation	40
4.5.2.1 FileCorruptError()	40
4.6 FileLockError Class Reference	41
4.6.1 Detailed Description	42
4.6.2 Constructor & Destructor Documentation	42
4.6.2.1 FileLockError()	42
4.7 FileNotFoundException Class Reference	42
4.7.1 Detailed Description	43
4.7.2 Constructor & Destructor Documentation	44

---

4.7.2.1 FileNotFoundException() . . . . .	44
4.8 FileOpenError Class Reference . . . . .	44
4.8.1 Detailed Description . . . . .	45
4.8.2 Constructor & Destructor Documentation . . . . .	45
4.8.2.1 FileOpenError() . . . . .	45
4.9 FilePermissionError Class Reference . . . . .	46
4.9.1 Detailed Description . . . . .	47
4.9.2 Constructor & Destructor Documentation . . . . .	47
4.9.2.1 FilePermissionError() . . . . .	47
4.10 FileReadError Class Reference . . . . .	47
4.10.1 Detailed Description . . . . .	48
4.10.2 Constructor & Destructor Documentation . . . . .	49
4.10.2.1 FileReadError() . . . . .	49
4.11 FileWriteError Class Reference . . . . .	49
4.11.1 Detailed Description . . . . .	50
4.11.2 Constructor & Destructor Documentation . . . . .	50
4.11.2.1 FileWriteError() . . . . .	50
4.12 GreenHouse Class Reference . . . . .	51
4.12.1 Detailed Description . . . . .	52
4.12.2 Constructor & Destructor Documentation . . . . .	53
4.12.2.1 GreenHouse() . . . . .	53
4.12.2.2 ~GreenHouse() . . . . .	53
4.12.3 Member Function Documentation . . . . .	53
4.12.3.1 mainSystem() . . . . .	54
4.12.3.2 manageAdmin() . . . . .	56
4.12.3.3 manageCreateUser() . . . . .	59
4.12.3.4 manageDeleteUser() . . . . .	60
4.12.3.5 manageEmployee() . . . . .	60
4.12.3.6 manageGuest() . . . . .	63
4.12.3.7 manageLogin() . . . . .	64
4.12.3.8 manageUpdateUser() . . . . .	67
4.12.3.9 save() . . . . .	68
4.12.3.10 searchUser() . . . . .	69
4.12.3.11 sleep() . . . . .	69
4.12.3.12 startSystem() . . . . .	70
4.12.4 Member Data Documentation . . . . .	70
4.12.4.1 alarm_ . . . . .	70
4.12.4.2 mc_ . . . . .	71
4.12.4.3 ms_ . . . . .	71
4.12.4.4 us_ . . . . .	71
4.13 Hardware Class Reference . . . . .	72
4.13.1 Detailed Description . . . . .	74

---

4.13.2 Member Enumeration Documentation . . . . .	74
4.13.2.1 Types_Hardware . . . . .	74
4.13.3 Constructor & Destructor Documentation . . . . .	74
4.13.3.1 Hardware() [1/2] . . . . .	74
4.13.3.2 Hardware() [2/2] . . . . .	75
4.13.3.3 ~Hardware() . . . . .	75
4.13.4 Member Function Documentation . . . . .	75
4.13.4.1 askInput() . . . . .	75
4.13.4.2 displayOutput() . . . . .	76
4.13.4.3 getType() . . . . .	76
4.13.4.4 isActive() . . . . .	77
4.13.4.5 setType() . . . . .	77
4.13.4.6 stringStatus() . . . . .	78
4.13.4.7 stringToType() . . . . .	79
4.13.4.8 turnOff() . . . . .	80
4.13.4.9 turnOn() . . . . .	80
4.13.4.10 typeToString() . . . . .	81
4.13.5 Member Data Documentation . . . . .	81
4.13.5.1 active_ . . . . .	81
4.13.5.2 type_ . . . . .	82
4.14 HydrometerSensor Class Reference . . . . .	82
4.14.1 Detailed Description . . . . .	84
4.14.2 Constructor & Destructor Documentation . . . . .	84
4.14.2.1 HydrometerSensor() . . . . .	84
4.14.2.2 ~HydrometerSensor() . . . . .	84
4.14.3 Member Function Documentation . . . . .	85
4.14.3.1 checkAllgood() . . . . .	85
4.14.3.2 collectAndPrint() . . . . .	86
4.14.3.3 collectData() . . . . .	86
4.14.3.4 printData() . . . . .	87
4.14.3.5 stringStatus() . . . . .	88
4.14.4 Friends And Related Function Documentation . . . . .	89
4.14.4.1 operator<< . . . . .	89
4.15 KeyboardHardware Class Reference . . . . .	90
4.15.1 Detailed Description . . . . .	92
4.15.2 Constructor & Destructor Documentation . . . . .	92
4.15.2.1 KeyboardHardware() . . . . .	92
4.15.2.2 ~KeyboardHardware() . . . . .	92
4.15.3 Member Function Documentation . . . . .	93
4.15.3.1 askInput() . . . . .	93
4.15.3.2 displayOutput() . . . . .	94
4.15.3.3 stringInput() . . . . .	94

---

4.15.4 Member Data Documentation . . . . .	95
4.15.4.1 data_ . . . . .	95
4.15.4.2 stringData_ . . . . .	95
4.16 LightSensor Class Reference . . . . .	95
4.16.1 Detailed Description . . . . .	98
4.16.2 Constructor & Destructor Documentation . . . . .	98
4.16.2.1 LightSensor() . . . . .	98
4.16.2.2 ~LightSensor() . . . . .	98
4.16.3 Member Function Documentation . . . . .	99
4.16.3.1 checkAllgood() . . . . .	99
4.16.3.2 collectAndPrint() . . . . .	100
4.16.3.3 collectData() . . . . .	100
4.16.3.4 printData() . . . . .	101
4.16.3.5 stringStatus() . . . . .	102
4.16.4 Friends And Related Function Documentation . . . . .	103
4.16.4.1 operator<< . . . . .	103
4.17 ManageCameras Class Reference . . . . .	103
4.17.1 Detailed Description . . . . .	105
4.17.2 Constructor & Destructor Documentation . . . . .	105
4.17.2.1 ManageCameras() . . . . .	105
4.17.2.2 ~ManageCameras() . . . . .	105
4.17.3 Member Function Documentation . . . . .	105
4.17.3.1 addCamera() . . . . .	105
4.17.3.2 clearCameras() . . . . .	106
4.17.3.3 collectAndDisplayData() . . . . .	106
4.17.3.4 collectData() . . . . .	107
4.17.3.5 createCamera() . . . . .	107
4.17.3.6 displayAllCameras() . . . . .	108
4.17.3.7 findCamera() . . . . .	108
4.17.3.8 loadCameras() . . . . .	109
4.17.3.9 removeCamera() . . . . .	110
4.17.3.10 saveCameras() . . . . .	111
4.17.3.11 turnOffCameras() . . . . .	112
4.17.3.12 turnOnCameras() . . . . .	113
4.17.3.13 turnOnOffCameras() . . . . .	113
4.17.4 Member Data Documentation . . . . .	113
4.17.4.1 cameras_ . . . . .	114
4.17.4.2 filename_ . . . . .	114
4.18 MonitoringSystem Class Reference . . . . .	114
4.18.1 Detailed Description . . . . .	118
4.18.2 Constructor & Destructor Documentation . . . . .	119
4.18.2.1 MonitoringSystem() . . . . .	119

4.18.2.2 ~MonitoringSystem()	119
4.18.3 Member Function Documentation	119
4.18.3.1 askEmail()	119
4.18.3.2 askIdAlarm()	120
4.18.3.3 askIdCamera()	121
4.18.3.4 askInputInt()	121
4.18.3.5 askMainUser()	123
4.18.3.6 askName()	123
4.18.3.7 askNIF()	124
4.18.3.8 askPassword()	125
4.18.3.9 askPrivileges()	126
4.18.3.10 askTypeAlarm()	127
4.18.3.11 askTypeCamera()	127
4.18.3.12 cleanScreen()	128
4.18.3.13 createAlarmScreen()	129
4.18.3.14 createCameraScreen()	130
4.18.3.15 createUserScreen()	131
4.18.3.16 deleteAlarmScreen()	131
4.18.3.17 deleteCameraScreen()	132
4.18.3.18 deleteUserScreen()	133
4.18.3.19 displayAlarmsScreen()	134
4.18.3.20 displayCameraScreen()	134
4.18.3.21 displayErrorScreen()	135
4.18.3.22 displaySensorsScreen()	136
4.18.3.23 displayUsersScreen()	137
4.18.3.24 exitScreen()	138
4.18.3.25 getEmailSelectedUser()	138
4.18.3.26 getIdAlarm()	139
4.18.3.27 getIdCamera()	140
4.18.3.28 getIdSelectedAlarm()	140
4.18.3.29 getIdSelectedCamera()	141
4.18.3.30 getName()	141
4.18.3.31 getNameSelectedUser()	142
4.18.3.32 getNIF()	142
4.18.3.33 getNIFSelectedUser()	143
4.18.3.34 getPassword()	143
4.18.3.35 getPasswordSelectedUser()	144
4.18.3.36 getPrivilegesSelectedUser()	144
4.18.3.37 getSelection()	145
4.18.3.38 getTypeAlarm()	145
4.18.3.39 getTypeCamera()	146
4.18.3.40 initialScreen()	146

---

4.18.3.41 inputCorrect() . . . . .	147
4.18.3.42 loginScreen() . . . . .	148
4.18.3.43 mainWindowAdmin() . . . . .	148
4.18.3.44 mainWindowEmployee() . . . . .	149
4.18.3.45 mainWindowGuest() . . . . .	150
4.18.3.46 saveAlarmScreen() . . . . .	151
4.18.3.47 saveCameraScreen() . . . . .	152
4.18.3.48 saveUsersScreen() . . . . .	152
4.18.3.49 selectUser() . . . . .	153
4.18.3.50 shortSelectUser() . . . . .	154
4.18.3.51 turnOnOffCameraScreen() . . . . .	155
4.18.3.52 turnOnOffSystemScreen() . . . . .	155
4.18.3.53 updateUserScreen() . . . . .	156
4.18.4 Member Data Documentation . . . . .	157
4.18.4.1 emailSelectedUser_ . . . . .	157
4.18.4.2 idAlarm_ . . . . .	157
4.18.4.3 idCamera_ . . . . .	157
4.18.4.4 idSelectedAlarm_ . . . . .	158
4.18.4.5 idSelectedCamera_ . . . . .	158
4.18.4.6 keyboard . . . . .	158
4.18.4.7 name_ . . . . .	158
4.18.4.8 nameSelectedUser_ . . . . .	159
4.18.4.9 nif_ . . . . .	159
4.18.4.10 nifSelectedUser_ . . . . .	159
4.18.4.11 password_ . . . . .	159
4.18.4.12 passwordSelectedUser_ . . . . .	160
4.18.4.13 privilegesSelectedUser_ . . . . .	160
4.18.4.14 screen . . . . .	160
4.18.4.15 selection_ . . . . .	160
4.18.4.16 sw . . . . .	161
4.18.4.17 typeAlarm_ . . . . .	161
4.18.4.18 typeCamera_ . . . . .	161
4.19 PhSensor Class Reference . . . . .	162
4.19.1 Detailed Description . . . . .	164
4.19.2 Constructor & Destructor Documentation . . . . .	164
4.19.2.1 PhSensor() . . . . .	164
4.19.2.2 ~PhSensor() . . . . .	165
4.19.3 Member Function Documentation . . . . .	165
4.19.3.1 checkAllgood() . . . . .	165
4.19.3.2 collectAndPrint() . . . . .	166
4.19.3.3 collectData() . . . . .	166
4.19.3.4 printData() . . . . .	167

4.19.3.5 qualifyPh() . . . . .	168
4.19.3.6 stringStatus() . . . . .	169
4.19.4 Friends And Related Function Documentation . . . . .	170
4.19.4.1 operator<< . . . . .	170
4.20 PressureSensor Class Reference . . . . .	170
4.20.1 Detailed Description . . . . .	173
4.20.2 Constructor & Destructor Documentation . . . . .	173
4.20.2.1 PressureSensor() . . . . .	173
4.20.2.2 ~PressureSensor() . . . . .	173
4.20.3 Member Function Documentation . . . . .	174
4.20.3.1 checkAllgood() . . . . .	174
4.20.3.2 collectAndPrint() . . . . .	175
4.20.3.3 collectData() . . . . .	175
4.20.3.4 printData() . . . . .	176
4.20.3.5 stringStatus() . . . . .	177
4.20.4 Friends And Related Function Documentation . . . . .	178
4.20.4.1 operator<< . . . . .	178
4.21 RGBCamera Class Reference . . . . .	179
4.21.1 Detailed Description . . . . .	181
4.21.2 Constructor & Destructor Documentation . . . . .	181
4.21.2.1 RGBCamera() . . . . .	181
4.21.2.2 ~RGBCamera() . . . . .	182
4.21.3 Member Function Documentation . . . . .	182
4.21.3.1 collectData() . . . . .	182
4.21.3.2 getBlue() . . . . .	183
4.21.3.3 getGreen() . . . . .	184
4.21.3.4 getRed() . . . . .	184
4.21.3.5 printCamera() . . . . .	185
4.21.3.6 randomComponent() . . . . .	186
4.21.3.7 setBlue() . . . . .	186
4.21.3.8 setGreen() . . . . .	187
4.21.3.9 setRed() . . . . .	187
4.21.3.10 setRGB() . . . . .	188
4.21.4 Member Data Documentation . . . . .	189
4.21.4.1 blue_ . . . . .	189
4.21.4.2 green_ . . . . .	189
4.21.4.3 red_ . . . . .	190
4.22 ScreenHardware Class Reference . . . . .	190
4.22.1 Detailed Description . . . . .	194
4.22.2 Constructor & Destructor Documentation . . . . .	194
4.22.2.1 ScreenHardware() . . . . .	194
4.22.2.2 ~ScreenHardware() . . . . .	194

---

4.22.3 Member Function Documentation . . . . .	194
4.22.3.1 askInput() . . . . .	195
4.22.3.2 cleanScreen() . . . . .	195
4.22.3.3 createAlarmWindow() . . . . .	196
4.22.3.4 createCameraWindow() . . . . .	197
4.22.3.5 createUserWindow() . . . . .	197
4.22.3.6 deleteAlarmWindow() . . . . .	198
4.22.3.7 deleteCameraWindow() . . . . .	198
4.22.3.8 deleteUserWindow() . . . . .	199
4.22.3.9 displayAlarmsWindow() . . . . .	199
4.22.3.10 displayCameraWindow() . . . . .	200
4.22.3.11 displayErrorWindow() . . . . .	200
4.22.3.12 displayOutput() . . . . .	201
4.22.3.13 displaySensorsWindow() . . . . .	201
4.22.3.14 displayUsersWindow() . . . . .	201
4.22.3.15 exitWindow() . . . . .	202
4.22.3.16 initialWindow() . . . . .	202
4.22.3.17 loginWindow() . . . . .	203
4.22.3.18 mainWindowAdmin() . . . . .	203
4.22.3.19 mainWindowEmployee() . . . . .	204
4.22.3.20 mainWindowGuest() . . . . .	205
4.22.3.21 saveAlarmWindow() . . . . .	205
4.22.3.22 saveCameraWindow() . . . . .	206
4.22.3.23 saveUsersWindow() . . . . .	206
4.22.3.24 turnOnOffCameraWindow() . . . . .	207
4.22.3.25 turnOnOffSystemWindow() . . . . .	207
4.22.3.26 updateUserWindow() . . . . .	208
4.23 Sensor Class Reference . . . . .	208
4.23.1 Detailed Description . . . . .	211
4.23.2 Member Enumeration Documentation . . . . .	211
4.23.2.1 Types . . . . .	211
4.23.3 Constructor & Destructor Documentation . . . . .	211
4.23.3.1 Sensor() [1/2] . . . . .	211
4.23.3.2 Sensor() [2/2] . . . . .	212
4.23.3.3 ~Sensor() . . . . .	212
4.23.4 Member Function Documentation . . . . .	212
4.23.4.1 checkAllgood() . . . . .	213
4.23.4.2 collectAndPrint() . . . . .	213
4.23.4.3 collectData() . . . . .	214
4.23.4.4 getData() . . . . .	214
4.23.4.5 getId() . . . . .	215
4.23.4.6 getType() . . . . .	216

---

4.23.4.7 isActive() . . . . .	217
4.23.4.8 operator<() . . . . .	217
4.23.4.9 operator==( ) . . . . .	218
4.23.4.10 operator>() . . . . .	218
4.23.4.11 printData() . . . . .	219
4.23.4.12 setData() . . . . .	219
4.23.4.13 setId() . . . . .	220
4.23.4.14 setType() . . . . .	221
4.23.4.15 stringToType() . . . . .	221
4.23.4.16 turnOff() . . . . .	222
4.23.4.17 turnOn() . . . . .	223
4.23.4.18 typeToString() . . . . .	223
4.23.5 Friends And Related Function Documentation . . . . .	224
4.23.5.1 operator<< . . . . .	224
4.23.5.2 operator>> . . . . .	224
4.23.6 Member Data Documentation . . . . .	225
4.23.6.1 active_ . . . . .	225
4.23.6.2 data_ . . . . .	225
4.23.6.3 id_ . . . . .	225
4.23.6.4 type_ . . . . .	225
4.24 SwitchHardware Class Reference . . . . .	226
4.24.1 Detailed Description . . . . .	228
4.24.2 Constructor & Destructor Documentation . . . . .	228
4.24.2.1 SwitchHardware() . . . . .	228
4.24.2.2 ~SwitchHardware() . . . . .	228
4.24.3 Member Function Documentation . . . . .	228
4.24.3.1 askInput() . . . . .	229
4.24.3.2 displayOutput() . . . . .	230
4.24.3.3 translateInput() . . . . .	230
4.25 TemperatureCamera Class Reference . . . . .	231
4.25.1 Detailed Description . . . . .	234
4.25.2 Constructor & Destructor Documentation . . . . .	234
4.25.2.1 TemperatureCamera() . . . . .	234
4.25.2.2 ~TemperatureCamera() . . . . .	235
4.25.3 Member Function Documentation . . . . .	235
4.25.3.1 collectData() . . . . .	235
4.25.3.2 getTemperature() . . . . .	236
4.25.3.3 printCamera() . . . . .	237
4.25.3.4 randomTemperature() . . . . .	238
4.25.3.5 setTemperature() . . . . .	238
4.25.4 Member Data Documentation . . . . .	239
4.25.4.1 temperature_ . . . . .	239

---

4.26 TemperatureSensor Class Reference . . . . .	239
4.26.1 Detailed Description . . . . .	242
4.26.2 Constructor & Destructor Documentation . . . . .	242
4.26.2.1 TemperatureSensor() . . . . .	242
4.26.2.2 ~TemperatureSensor() . . . . .	242
4.26.3 Member Function Documentation . . . . .	243
4.26.3.1 checkAllgood() . . . . .	243
4.26.3.2 collectAndPrint() . . . . .	244
4.26.3.3 collectData() . . . . .	244
4.26.3.4 printData() . . . . .	245
4.26.3.5 stringStatus() . . . . .	246
4.26.4 Friends And Related Function Documentation . . . . .	247
4.26.4.1 operator<< . . . . .	247
4.27 User Class Reference . . . . .	248
4.27.1 Detailed Description . . . . .	250
4.27.2 Constructor & Destructor Documentation . . . . .	250
4.27.2.1 User() [1/2] . . . . .	251
4.27.2.2 User() [2/2] . . . . .	251
4.27.2.3 ~User() . . . . .	252
4.27.3 Member Function Documentation . . . . .	252
4.27.3.1 getEmail() . . . . .	253
4.27.3.2 getName() . . . . .	253
4.27.3.3 getNif() . . . . .	254
4.27.3.4 getPassword() . . . . .	255
4.27.3.5 getPrivileges() . . . . .	255
4.27.3.6 operator<() . . . . .	256
4.27.3.7 operator==() . . . . .	256
4.27.3.8 operator>() . . . . .	256
4.27.3.9 printUser() . . . . .	257
4.27.3.10 setEmail() . . . . .	257
4.27.3.11 setName() . . . . .	258
4.27.3.12 setNif() . . . . .	259
4.27.3.13 setPassword() . . . . .	260
4.27.3.14 setPrivileges() . . . . .	261
4.27.4 Friends And Related Function Documentation . . . . .	262
4.27.4.1 operator<< . . . . .	262
4.27.4.2 operator>> . . . . .	262
4.27.5 Member Data Documentation . . . . .	263
4.27.5.1 email . . . . .	263
4.27.5.2 name . . . . .	263
4.27.5.3 nif . . . . .	264
4.27.5.4 password . . . . .	264

---

4.27.5.5 privileges . . . . .	264
4.28 UserAdmin Class Reference . . . . .	265
4.28.1 Detailed Description . . . . .	266
4.28.2 Constructor & Destructor Documentation . . . . .	266
4.28.2.1 UserAdmin() [1/2] . . . . .	267
4.28.2.2 UserAdmin() [2/2] . . . . .	267
4.28.2.3 ~UserAdmin() . . . . .	268
4.29 UserEmployee Class Reference . . . . .	269
4.29.1 Detailed Description . . . . .	270
4.29.2 Constructor & Destructor Documentation . . . . .	270
4.29.2.1 UserEmployee() [1/2] . . . . .	271
4.29.2.2 UserEmployee() [2/2] . . . . .	271
4.29.2.3 ~UserEmployee() . . . . .	272
4.30 UserGuest Class Reference . . . . .	273
4.30.1 Detailed Description . . . . .	274
4.30.2 Constructor & Destructor Documentation . . . . .	274
4.30.2.1 UserGuest() [1/2] . . . . .	275
4.30.2.2 UserGuest() [2/2] . . . . .	275
4.30.2.3 ~UserGuest() . . . . .	276
4.31 UserNameComparator Class Reference . . . . .	277
4.31.1 Detailed Description . . . . .	277
4.31.2 Member Function Documentation . . . . .	277
4.31.2.1 operator()() . . . . .	277
4.32 UserPtrComparator Class Reference . . . . .	278
4.32.1 Detailed Description . . . . .	278
4.32.2 Member Function Documentation . . . . .	278
4.32.2.1 operator()() . . . . .	278
4.33 UsersDatabase Class Reference . . . . .	279
4.33.1 Detailed Description . . . . .	280
4.33.2 Constructor & Destructor Documentation . . . . .	280
4.33.2.1 UsersDatabase() . . . . .	281
4.33.2.2 ~UsersDatabase() . . . . .	281
4.33.3 Member Function Documentation . . . . .	281
4.33.3.1 addUser() . . . . .	281
4.33.3.2 deleteUser() . . . . .	282
4.33.3.3 deleteUserByEmail() . . . . .	283
4.33.3.4 deleteUserByName() . . . . .	284
4.33.3.5 deleteUserByNif() . . . . .	285
4.33.3.6 findUser() . . . . .	286
4.33.3.7 findUserByEmail() . . . . .	287
4.33.3.8 findUserByName() . . . . .	288
4.33.3.9 findUserByNif() . . . . .	289

---

4.33.3.10 findUserByPassword()	290
4.33.3.11 getUsers()	290
4.33.3.12 getUsersSortedByName()	291
4.33.3.13 isValidPrivileges()	291
4.33.3.14 printDeletedUser()	292
4.33.3.15 printUsers()	293
4.33.3.16 setUsers()	294
4.33.4 Member Data Documentation	294
4.33.4.1 users_	295
4.34 UsersServer Class Reference	295
4.34.1 Detailed Description	297
4.34.2 Constructor & Destructor Documentation	297
4.34.2.1 UsersServer()	298
4.34.2.2 ~UsersServer()	298
4.34.3 Member Function Documentation	298
4.34.3.1 createUser()	298
4.34.3.2 deleteUser()	299
4.34.3.3 findUserLogin()	299
4.34.3.4 getPrivileges()	300
4.34.3.5 loadUsersFromFile()	301
4.34.3.6 printUsersServer()	302
4.34.3.7 readUsersFromFileBi()	303
4.34.3.8 readUsersFromFileTxt()	305
4.34.3.9 saveUsersToFile()	305
4.34.3.10 updateUser()	306
4.34.3.11 writeUserToFileBi()	307
4.34.3.12 writeUserToFileTxt()	308
4.34.4 Member Data Documentation	308
4.34.4.1 fileNameBi_	308
4.34.4.2 fileNameTxt_	309
4.34.4.3 usersDatabase_	309
<b>5 File Documentation</b>	<b>311</b>
5.1 src/AirQualitySensor.cpp File Reference	311
5.1.1 Function Documentation	311
5.1.1.1 operator<<()	312
5.2 src/AirQualitySensor.h File Reference	312
5.2.1 Detailed Description	313
5.3 src/AlarmSensors.cpp File Reference	313
5.4 src/AlarmSensors.h File Reference	313
5.4.1 Detailed Description	314
5.5 src/Camera.cpp File Reference	315

---

5.6 src/Camera.h File Reference . . . . .	315
5.6.1 Detailed Description . . . . .	316
5.7 src/Exceptions.h File Reference . . . . .	316
5.7.1 Detailed Description . . . . .	317
5.8 src/GreenHouse.cpp File Reference . . . . .	317
5.9 src/GreenHouse.h File Reference . . . . .	318
5.9.1 Detailed Description . . . . .	319
5.10 src/Hardware.cpp File Reference . . . . .	319
5.11 src/Hardware.h File Reference . . . . .	320
5.11.1 Detailed Description . . . . .	320
5.12 src/HydrometerSensor.cpp File Reference . . . . .	321
5.12.1 Function Documentation . . . . .	321
5.12.1.1 operator<<() . . . . .	321
5.13 src/HydrometerSensor.h File Reference . . . . .	322
5.13.1 Detailed Description . . . . .	322
5.14 src/KeyboardHardware.cpp File Reference . . . . .	323
5.15 src/KeyboardHardware.h File Reference . . . . .	323
5.15.1 Detailed Description . . . . .	324
5.16 src/LightSensor.cpp File Reference . . . . .	324
5.16.1 Function Documentation . . . . .	325
5.16.1.1 operator<<() . . . . .	325
5.17 src/LightSensor.h File Reference . . . . .	325
5.17.1 Detailed Description . . . . .	326
5.18 src/main.cpp File Reference . . . . .	326
5.18.1 Function Documentation . . . . .	327
5.18.1.1 main() . . . . .	327
5.19 src/mainAirQualitySensor.cpp File Reference . . . . .	328
5.19.1 Function Documentation . . . . .	328
5.19.1.1 main() . . . . .	328
5.20 src/mainAlarmSensors.cpp File Reference . . . . .	329
5.20.1 Function Documentation . . . . .	329
5.20.1.1 main() . . . . .	330
5.21 src/mainGreenHouse.cpp File Reference . . . . .	330
5.21.1 Function Documentation . . . . .	331
5.21.1.1 main() . . . . .	331
5.22 src/mainHardware.cpp File Reference . . . . .	332
5.22.1 Function Documentation . . . . .	333
5.22.1.1 main() . . . . .	333
5.23 src/mainHydrometerSensor.cpp File Reference . . . . .	334
5.23.1 Function Documentation . . . . .	334
5.23.1.1 main() . . . . .	335
5.24 src/mainKeyboardHardware.cpp File Reference . . . . .	335

---

5.24.1 Function Documentation . . . . .	336
5.24.1.1 main() . . . . .	336
5.25 src/mainLightSensor.cpp File Reference . . . . .	337
5.25.1 Function Documentation . . . . .	338
5.25.1.1 main() . . . . .	338
5.26 src/mainManageCameras.cpp File Reference . . . . .	338
5.26.1 Function Documentation . . . . .	339
5.26.1.1 main() . . . . .	339
5.27 src/mainMonitoringSystem.cpp File Reference . . . . .	340
5.27.1 Function Documentation . . . . .	341
5.27.1.1 main() . . . . .	341
5.28 src/mainPhSensor.cpp File Reference . . . . .	342
5.28.1 Function Documentation . . . . .	343
5.28.1.1 main() . . . . .	343
5.29 src/mainPressureSensor.cpp File Reference . . . . .	344
5.29.1 Function Documentation . . . . .	344
5.29.1.1 main() . . . . .	344
5.30 src/mainRGBCamera.cpp File Reference . . . . .	345
5.30.1 Function Documentation . . . . .	345
5.30.1.1 main() . . . . .	346
5.31 src/mainScreenHardware.cpp File Reference . . . . .	346
5.31.1 Function Documentation . . . . .	347
5.31.1.1 main() . . . . .	347
5.32 src/mainSensor.cpp File Reference . . . . .	348
5.32.1 Function Documentation . . . . .	349
5.32.1.1 main() . . . . .	349
5.33 src/mainSwitchHardware.cpp File Reference . . . . .	351
5.33.1 Function Documentation . . . . .	352
5.33.1.1 main() . . . . .	352
5.34 src/mainTemperatureCamera.cpp File Reference . . . . .	353
5.34.1 Function Documentation . . . . .	353
5.34.1.1 main() . . . . .	354
5.35 src/mainTemperatureSensor.cpp File Reference . . . . .	354
5.35.1 Function Documentation . . . . .	355
5.35.1.1 main() . . . . .	355
5.36 src/mainUser.cpp File Reference . . . . .	356
5.36.1 Function Documentation . . . . .	356
5.36.1.1 main() . . . . .	357
5.37 src/mainUserAdmin.cpp File Reference . . . . .	358
5.37.1 Function Documentation . . . . .	359
5.37.1.1 main() . . . . .	359
5.38 src/mainUserEmployee.cpp File Reference . . . . .	360

5.38.1 Function Documentation . . . . .	361
5.38.1.1 main() . . . . .	361
5.39 src/mainUserGuest.cpp File Reference . . . . .	362
5.39.1 Function Documentation . . . . .	362
5.39.1.1 main() . . . . .	362
5.40 src/mainUsersDatabase.cpp File Reference . . . . .	363
5.40.1 Function Documentation . . . . .	364
5.40.1.1 main() . . . . .	364
5.41 src/mainUsersServer.cpp File Reference . . . . .	365
5.41.1 Function Documentation . . . . .	366
5.41.1.1 main() . . . . .	366
5.42 src/ManageCameras.cpp File Reference . . . . .	367
5.43 src/ManageCameras.h File Reference . . . . .	368
5.43.1 Detailed Description . . . . .	369
5.44 src/MonitoringSystem.cpp File Reference . . . . .	370
5.44.1 Variable Documentation . . . . .	370
5.44.1.1 ADMIN_MENU_OPTIONS . . . . .	370
5.44.1.2 EMPLOYEE_MENU_OPTIONS . . . . .	370
5.44.1.3 GUEST_MENU_OPTIONS . . . . .	371
5.44.1.4 MAIN_MENU_OPTIONS . . . . .	371
5.45 src/MonitoringSystem.h File Reference . . . . .	371
5.45.1 Detailed Description . . . . .	372
5.45.2 Variable Documentation . . . . .	372
5.45.2.1 ADMIN_MENU_OPTIONS . . . . .	372
5.45.2.2 EMPLOYEE_MENU_OPTIONS . . . . .	372
5.45.2.3 GUEST_MENU_OPTIONS . . . . .	373
5.45.2.4 MAIN_MENU_OPTIONS . . . . .	373
5.46 src/PhSensor.cpp File Reference . . . . .	373
5.46.1 Function Documentation . . . . .	374
5.46.1.1 operator<<() . . . . .	374
5.47 src/PhSensor.h File Reference . . . . .	374
5.47.1 Detailed Description . . . . .	375
5.48 src/PressureSensor.cpp File Reference . . . . .	375
5.48.1 Function Documentation . . . . .	376
5.48.1.1 operator<<() . . . . .	376
5.49 src/PressureSensor.h File Reference . . . . .	376
5.49.1 Detailed Description . . . . .	377
5.50 src/RGBCamera.cpp File Reference . . . . .	377
5.51 src/RGBCamera.h File Reference . . . . .	378
5.51.1 Detailed Description . . . . .	378
5.52 src/ScreenHardware.cpp File Reference . . . . .	379
5.52.1 Variable Documentation . . . . .	379

---

5.52.1.1 ASK_DATA . . . . .	380
5.52.1.2 USER_PROMPT . . . . .	380
5.53 src/ScreenHardware.h File Reference . . . . .	380
5.53.1 Detailed Description . . . . .	381
5.54 src/Sensor.cpp File Reference . . . . .	381
5.54.1 Function Documentation . . . . .	381
5.54.1.1 operator<<() . . . . .	382
5.54.1.2 operator>>() . . . . .	383
5.55 src/Sensor.h File Reference . . . . .	383
5.55.1 Detailed Description . . . . .	384
5.56 src/SwitchHardware.cpp File Reference . . . . .	384
5.57 src/SwitchHardware.h File Reference . . . . .	385
5.57.1 Detailed Description . . . . .	386
5.58 src/TemperatureCamera.cpp File Reference . . . . .	386
5.59 src/TemperatureCamera.h File Reference . . . . .	387
5.59.1 Detailed Description . . . . .	387
5.60 src/TemperatureSensor.cpp File Reference . . . . .	388
5.60.1 Function Documentation . . . . .	388
5.60.1.1 operator<<() . . . . .	388
5.61 src/TemperatureSensor.h File Reference . . . . .	389
5.61.1 Detailed Description . . . . .	389
5.62 src/User.cpp File Reference . . . . .	390
5.62.1 Function Documentation . . . . .	390
5.62.1.1 operator<<() . . . . .	390
5.62.1.2 operator>>() . . . . .	391
5.63 src/User.h File Reference . . . . .	391
5.63.1 Detailed Description . . . . .	392
5.64 src/UserAdmin.cpp File Reference . . . . .	392
5.65 src/UserAdmin.h File Reference . . . . .	393
5.65.1 Detailed Description . . . . .	393
5.66 src/UserEmployee.cpp File Reference . . . . .	394
5.67 src/UserEmployee.h File Reference . . . . .	394
5.67.1 Detailed Description . . . . .	395
5.68 src/UserGuest.cpp File Reference . . . . .	395
5.69 src/UserGuest.h File Reference . . . . .	396
5.69.1 Detailed Description . . . . .	396
5.70 src/UsersDatabase.cpp File Reference . . . . .	397
5.71 src/UsersDatabase.h File Reference . . . . .	397
5.71.1 Detailed Description . . . . .	398
5.72 src/UsersServer.cpp File Reference . . . . .	398
5.73 src/UsersServer.h File Reference . . . . .	399
5.73.1 Detailed Description . . . . .	400



# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

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

AlarmSensors . . . . .	18
Camera . . . . .	30
RGBCamera . . . . .	179
TemperatureCamera . . . . .	231
GreenHouse . . . . .	51
Hardware . . . . .	72
KeyboardHardware . . . . .	90
ScreenHardware . . . . .	190
SwitchHardware . . . . .	226
ManageCameras . . . . .	103
MonitoringSystem . . . . .	114
std::runtime_error . . . . .	
FileCloseError . . . . .	38
FileCorruptError . . . . .	39
FileLockError . . . . .	41
FileNotFoundException . . . . .	42
FileOpenError . . . . .	44
FilePermissionError . . . . .	46
FileReadError . . . . .	47
FileWriteError . . . . .	49
Sensor . . . . .	208
AirQualitySensor . . . . .	9
HydrometerSensor . . . . .	82
LightSensor . . . . .	95
PhSensor . . . . .	162
PressureSensor . . . . .	170
TemperatureSensor . . . . .	239
User . . . . .	248
UserAdmin . . . . .	265
UserEmployee . . . . .	269
UserGuest . . . . .	273
UserNameComparator . . . . .	277
UserPtrComparator . . . . .	278
UsersDatabase . . . . .	279
UsersServer . . . . .	295



# Chapter 2

## Class Index

### 2.1 Class List

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

AirQualitySensor . . . . .	9
AlarmSensors . . . . .	18
Camera . . . . .	30
FileCloseError . . . . .	38
FileCorruptError . . . . .	39
FileLockError . . . . .	41
FileNotFoundException . . . . .	42
FileOpenError . . . . .	44
FilePermissionError . . . . .	46
FileReadError . . . . .	47
FileWriteError . . . . .	49
GreenHouse . . . . .	51
Hardware . . . . .	72
HydrometerSensor . . . . .	82
KeyboardHardware . . . . .	90
LightSensor . . . . .	95
ManageCameras . . . . .	103
MonitoringSystem . . . . .	114
PhSensor . . . . .	162
PressureSensor . . . . .	170
RGBCamera . . . . .	179
ScreenHardware . . . . .	190
Sensor . . . . .	208
SwitchHardware . . . . .	226
TemperatureCamera . . . . .	231
TemperatureSensor . . . . .	239
User . . . . .	248
UserAdmin . . . . .	265
UserEmployee . . . . .	269
UserGuest . . . . .	273
UserNameComparator . . . . .	277
UserPtrComparator . . . . .	278
UsersDatabase . . . . .	279
UsersServer . . . . .	295



# Chapter 3

## File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

src/AirQualitySensor.cpp . . . . .	311
src/AirQualitySensor.h	
This is the class <a href="#">AirQualitySensor</a> . It contains the attributes and methods of the <a href="#">AirQualitySensor</a> class . . . . .	312
src/AlarmSensors.cpp . . . . .	313
src/AlarmSensors.h	
This is the class <a href="#">AlarmSensors</a> . It contains the attributes and methods of the <a href="#">AlarmSensors</a> class . . . . .	313
src/Camera.cpp . . . . .	315
src/Camera.h	
This is the class <a href="#">Camera</a> . It contains the attributes and methods of the <a href="#">Camera</a> class. This class is used to represent a camera of the system, it can collect data, turn on/off the camera and print the camera information . . . . .	315
src/Exceptions.h	
This file contains the attributes and methods of the Exceptions class . . . . .	316
src/GreenHouse.cpp . . . . .	317
src/GreenHouse.h	
This is the class <a href="#">GreenHouse</a> . It contains the attributes and methods of the <a href="#">GreenHouse</a> class, this class is the main of the hole system . . . . .	318
src/Hardware.cpp . . . . .	319
src/Hardware.h	
This is the class <a href="#">Hardware</a> . It contains the attributes and methods of the <a href="#">Hardware</a> class, this class is the parent of the hole hardware system . . . . .	320
src/HydrometerSensor.cpp . . . . .	321
src/HydrometerSensor.h	
This is the class <a href="#">HydrometerSensor</a> . It contains the attributes and methods of the <a href="#">HydrometerSensor</a> class . . . . .	322
src/KeyboardHardware.cpp . . . . .	323
src/KeyboardHardware.h	
This is the class <a href="#">KeyboardHardware</a> . It contains the attributes and methods of the <a href="#">KeyboardHardware</a> class, this class is a child of the <a href="#">Hardware</a> class . . . . .	323
src/LightSensor.cpp . . . . .	324
src/LightSensor.h	
This is the class <a href="#">LightSensor</a> . It contains the attributes and methods of the <a href="#">LightSensor</a> class . . . . .	325
src/main.cpp . . . . .	326
src/mainAirQualitySensor.cpp . . . . .	328

src/mainAlarmSensors.cpp . . . . .	329
src/mainGreenHouse.cpp . . . . .	330
src/mainHardware.cpp . . . . .	332
src/mainHydrometerSensor.cpp . . . . .	334
src/mainKeyboardHardware.cpp . . . . .	335
src/mainLightSensor.cpp . . . . .	337
src/mainManageCameras.cpp . . . . .	338
src/mainMonitoringSystem.cpp . . . . .	340
src/mainPhSensor.cpp . . . . .	342
src/mainPressureSensor.cpp . . . . .	344
src/mainRGBCamera.cpp . . . . .	345
src/mainScreenHardware.cpp . . . . .	346
src/mainSensor.cpp . . . . .	348
src/mainSwitchHardware.cpp . . . . .	351
src/mainTemperatureCamera.cpp . . . . .	353
src/mainTemperatureSensor.cpp . . . . .	354
src/mainUser.cpp . . . . .	356
src/mainUserAdmin.cpp . . . . .	358
src/mainUserEmployee.cpp . . . . .	360
src/mainUserGuest.cpp . . . . .	362
src/mainUsersDatabase.cpp . . . . .	363
src/mainUsersServer.cpp . . . . .	365
src/ManageCameras.cpp . . . . .	367
src/ManageCameras.h . . . . .	
This is the class <a href="#">ManageCameras</a> . It contains the attributes and methods of the <a href="#">ManageCameras</a> class. This class is used to manage the cameras of the system, it can create, remove, display, collect data, turn on/off the cameras and save/load the cameras to/from a file . . . . .	368
src/MonitoringSystem.cpp . . . . .	370
src/MonitoringSystem.h . . . . .	
This is the class <a href="#">MonitoringSystem</a> . It contains the attributes and methods of the <a href="#">MonitoringSystem</a> class, this class . . . . .	371
src/PhSensor.cpp . . . . .	373
src/PhSensor.h . . . . .	
This is the class <a href="#">PhSensor</a> . It contains the attributes and methods of the <a href="#">PhSensor</a> class . . . . .	374
src/PressureSensor.cpp . . . . .	375
src/PressureSensor.h . . . . .	
This is the class <a href="#">PressureSensor</a> . It contains the attributes and methods of the <a href="#">PressureSensor</a> class . . . . .	376
src/RGBCamera.cpp . . . . .	377
src/RGBCamera.h . . . . .	
This is the class <a href="#">RGBCamera</a> . It contains the attributes and methods of the <a href="#">RGBCamera</a> class. This class is used to represent a RGB camera of the system, it can collect data, turn on/off the camera, print the camera information and set the RGB values of the camera . . . . .	378
src/ScreenHardware.cpp . . . . .	379
src/ScreenHardware.h . . . . .	
This is the class <a href="#">ScreenHardware</a> . It contains the attributes and methods of the <a href="#">ScreenHardware</a> class, this class is a child of the <a href="#">Hardware</a> class. This class is used to display the output of the system and ask for an input before with the keyboard . . . . .	380
src/Sensor.cpp . . . . .	381
src/Sensor.h . . . . .	
This is the class <a href="#">Sensor</a> . It contains the attributes and methods of the <a href="#">Sensor</a> class . . . . .	383
src/SwitchHardware.cpp . . . . .	384
src/SwitchHardware.h . . . . .	
This is the class <a href="#">SwitchHardware</a> . It contains the attributes and methods of the <a href="#">SwitchHardware</a> class, this class is a child of the <a href="#">Hardware</a> class . . . . .	385
src/TemperatureCamera.cpp . . . . .	386

src/TemperatureCamera.h	
This is the class Temperature Camera. It contains the attributes and methods of the TemperatureCamera class. This class is used to represent a temperature camera of the system, it can collect data, print the camera information and set the temperature of the camera	387
src/TemperatureSensor.cpp . . . . .	388
src/TemperatureSensor.h	
This is the class TemperatureSensor. It contains the attributes and methods of the TemperatureSensor class . . . . .	389
src/User.cpp . . . . .	390
src/User.h	
This is the class User. It contains the attributes and methods of the User class . . . . .	391
src/UserAdmin.cpp . . . . .	392
src/UserAdmin.h	
This is the class UserAdmin. It contains the attributes and methods of the UserAdmin class . . . . .	393
src/UserEmployee.cpp . . . . .	394
src/UserEmployee.h	
This is the class UserEmployee. It contains the attributes and methods of the UserEmployee class . . . . .	394
src/UserGuest.cpp . . . . .	395
src/UserGuest.h	
This is the class UserGuest. It contains the attributes and methods of the UserGuest class . . . . .	396
src/UsersDatabase.cpp . . . . .	397
src/UsersDatabase.h	
This is the class UsersDatabase. It contains the attributes and methods of the UsersDatabase class . . . . .	397
src/UsersServer.cpp . . . . .	398
src/UsersServer.h	
This is the class UsersServer. It contains the attributes and methods of the UsersServer class . . . . .	399



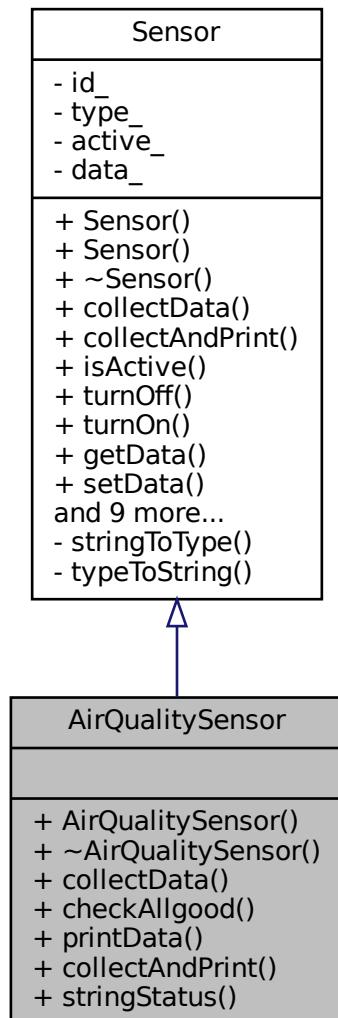
# **Chapter 4**

## **Class Documentation**

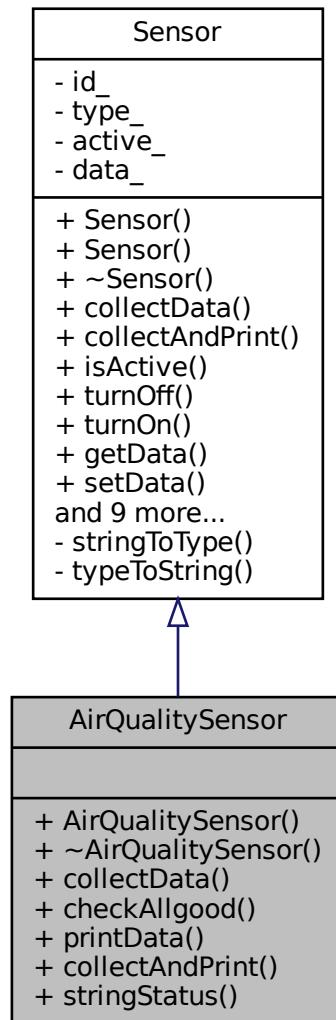
### **4.1 AirQualitySensor Class Reference**

```
#include <AirQualitySensor.h>
```

Inheritance diagram for AirQualitySensor:



Collaboration diagram for AirQualitySensor:



## Public Member Functions

- **AirQualitySensor** (int id, bool active)
 

*Construct a new Air Quality Sensor object.*
- **~AirQualitySensor** () override
 

*Destroy the Air Quality Sensor object.*
- void **collectData** () override
 

*Collect data of the Air Quality Sensor.*
- bool **checkAllgood** () const override
 

*Check if the Air Quality Sensor is working properly.*
- void **printData** () const override
 

*Print the data of the Air Quality Sensor.*
- void **collectAndPrint** ()
 

*Print the data of the Air Quality Sensor.*

*Collect and print the data of the Air Quality Sensor.*

- std::string [stringStatus \(\) const](#)

*This method returns if the Air Quality Sensor is active or not and if its active, it returns if its good or bad the data.*

## Friends

- std::ostream & [operator<< \(std::ostream &os, const AirQualitySensor &sensor\)](#)

*This method prints the AirQualitySensor object.*

## Additional Inherited Members

### 4.1.1 Detailed Description

Definition at line 15 of file AirQualitySensor.h.

### 4.1.2 Constructor & Destructor Documentation

#### 4.1.2.1 AirQualitySensor()

```
AirQualitySensor::AirQualitySensor (
    int id,
    bool active ) [explicit]
```

Construct a new Air Quality Sensor object.

##### Parameters

<i>id</i>	
<i>active</i>	

##### Returns

[AirQualitySensor](#) object

Definition at line 9 of file AirQualitySensor.cpp.

```
10     : Sensor(id, Sensor::Types::AIR_QUALITY, active) {}
```

#### 4.1.2.2 ~AirQualitySensor()

```
AirQualitySensor::~AirQualitySensor () [override]
```

Destroy the Air Quality Sensor object.

Definition at line 12 of file AirQualitySensor.cpp.

```
12 { }
```

### 4.1.3 Member Function Documentation

#### 4.1.3.1 checkAllgood()

```
bool AirQualitySensor::checkAllgood ( ) const [override], [virtual]
```

Check if the Air Quality [Sensor](#) is working properly.

##### Returns

true if the Air Quality [Sensor](#) is working properly  
false if the Air Quality [Sensor](#) is not working properly

Reimplemented from [Sensor](#).

Definition at line 24 of file [AirQualitySensor.cpp](#).

```
24     // Por debajo de 65 microgramos/m3 se considera buena calidad del aire
25     float data = Sensor::getData\(\);
26
27     if (data <= 65) {
28         return true;
29     } else {
30         return false;
31     }
32 }
```

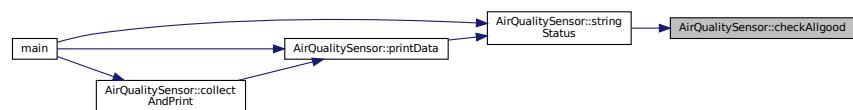
References [Sensor::getData\(\)](#).

Referenced by [stringStatus\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.3.2 collectAndPrint()

```
void AirQualitySensor::collectAndPrint ( ) [virtual]
```

Collect and print the data of the Air Quality [Sensor](#).

Reimplemented from [Sensor](#).

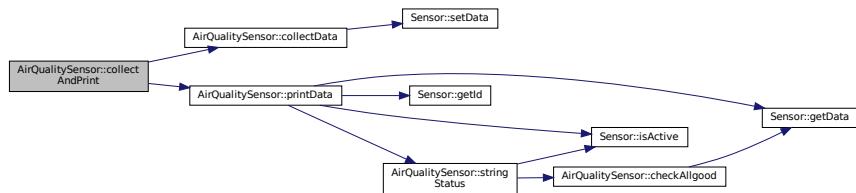
Definition at line 65 of file `AirQualitySensor.cpp`.

```
65 {  
66     collectData();  
67     printData();  
68 }
```

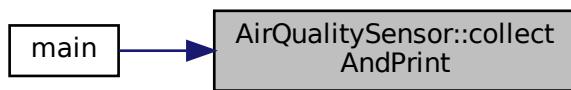
References `collectData()`, and `printData()`.

Referenced by `main()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.3.3 collectData()

```
void AirQualitySensor::collectData ( ) [override], [virtual]
```

Collect data of the Air Quality [Sensor](#).

This method collects the data of the Air Quality [Sensor](#) and stores it in the data attribute.

Reimplemented from [Sensor](#).

Definition at line 14 of file AirQualitySensor.cpp.

```

14      {
15      // Generamos un numero random entre 0 y 70 para simular la calidad del aire en
16      // microgramos/m3
17      std::random_device rd;
18      std::mt19937 gen(rd());
19      std::uniform_int_distribution<> dis(0, 70);
20      int airQuality = dis(gen);
21      Sensor::setData(airQuality);
22  }
```

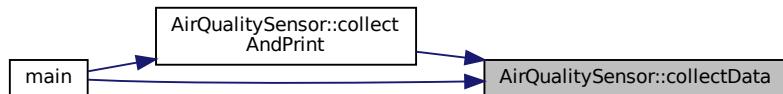
References Sensor::setData().

Referenced by collectAndPrint(), and main().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.3.4 printData()

```
void AirQualitySensor::printData ( ) const [override], [virtual]
```

Print the data of the Air Quality [Sensor](#).

Reimplemented from [Sensor](#).

Definition at line 52 of file AirQualitySensor.cpp.

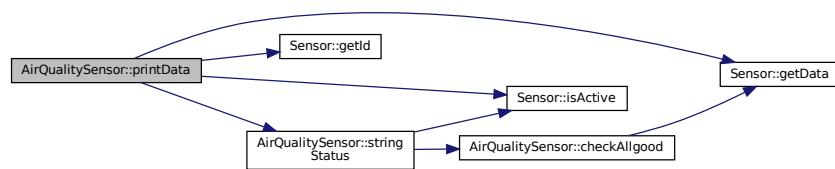
```

52      {
53      // Imprimimos las particulas por microgramo/m3, el id del sensor, y si todo
54      // esta bien o no
55      if (Sensor::isActive()) {
56          std::cout << "Air Quality Sensor with "
57          << "ID: " << Sensor::getId() << " - Data: " << Sensor::getData()
58          << " microgram/m3 - Status: " << stringStatus() << std::endl;
59      } else {
60          std::cout << "Air Quality Sensor ID: " << Sensor::getId() << " - INACTIVE"
61          << std::endl;
62      }
63  }
```

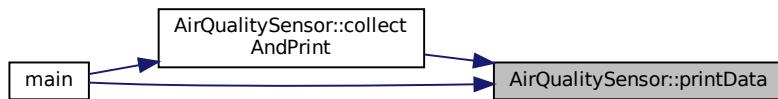
References Sensor::getData(), Sensor::getId(), Sensor::isActive(), and stringStatus().

Referenced by collectAndPrint(), and main().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.3.5 stringStatus()

```
std::string AirQualitySensor::stringStatus ( ) const
```

This method returns if the Air Quality Sensor is active or not and if its active, it returns if its good or bad the data.

Returns

std::string

Definition at line 40 of file AirQualitySensor.cpp.

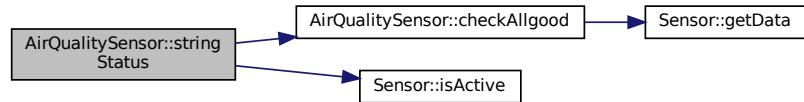
```

40
41     if (Sensor::isActive()) {
42         if (this->checkAllgood()) {
43             return "ACTIVE - GOOD STATUS";
44         } else {
45             return "ACTIVE - BAD STATUS";
46         }
47     } else {
48         return "INACTIVE";
49     }
50 }
```

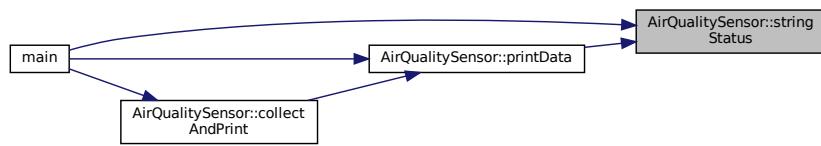
References checkAllgood(), and Sensor::isActive().

Referenced by main(), and printData().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.4 Friends And Related Function Documentation

##### 4.1.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const AirQualitySensor & sensor ) [friend]
```

This method prints the [AirQualitySensor](#) object.

###### Parameters

<i>os</i>	
<i>sensor</i>	

###### Returns

```
std::ostream&
```

Definition at line 35 of file [AirQualitySensor.cpp](#).

```
35
36     sensor.printData();
37     return os;
38 }
```

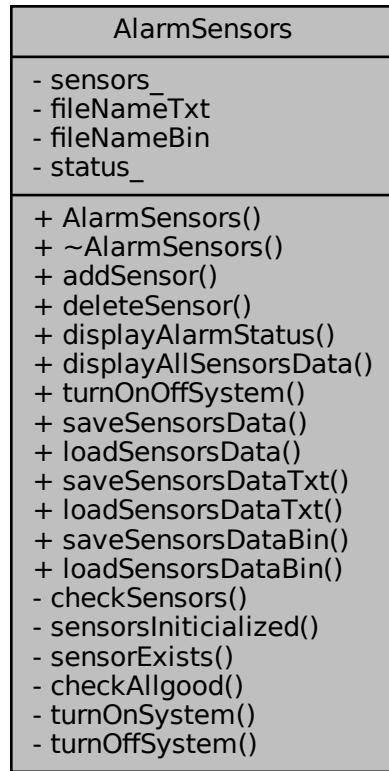
The documentation for this class was generated from the following files:

- [src/AirQualitySensor.h](#)
- [src/AirQualitySensor.cpp](#)

## 4.2 AlarmSensors Class Reference

```
#include <AlarmSensors.h>
```

Collaboration diagram for AlarmSensors:



### Public Member Functions

- [AlarmSensors \(\)](#)  
*Construct a new Alarm Sensors object.*
- [~AlarmSensors \(\)](#)  
*Destroy the Alarm Sensors object.*
- [void addSensor \(int id, std::string type\)](#)  
*Add a Sensor object.*
- [void deleteSensor \(int id\)](#)  
*Delete a Sensor object.*
- [void displayAlarmStatus \(\)](#)  
*Display the Alarm Status.*
- [void displayAllSensorsData \(\)](#)  
*Display all Sensors Data.*
- [void turnOnOffSystem \(int input\)](#)

*This method turns on or off the system.*

- void `saveSensorsData ()`

*This method saves the sensors data to a file, one .txt and other one .dat.*

- void `loadSensorsData ()`

*This method loads the sensors data from a file .dat, but you can change to loads the sensor from a .txt.*

- void `saveSensorsDataTxt ()`

*This method saves the sensors data to a file .txt.*

- void `loadSensorsDataTxt ()`

*This method loads the sensors data from a file .txt.*

- void `saveSensorsDataBin ()`

*This method saves the sensors data to a file .dat.*

- void `loadSensorsDataBin ()`

*This method loads the sensors data from a file .dat.*

## Private Member Functions

- int `checkSensors ()`

*Check the Sensors.*

- bool `sensorsInitialized ()`

*Check if the Sensors are Initialized.*

- bool `sensorExists (int id)`

*Check if a Sensor exists.*

- bool `checkAllgood ()`

*Check if the Sensors have good measurements.*

- void `turnOnSystem ()`

*Turn on the System.*

- void `turnOffSystem ()`

*Turn off the System.*

## Private Attributes

- std::set< `Sensor *` > `sensors_`

*This is the set of Sensor pointers.*

- std::string `fileNameTxt` = "sensors.txt"

*This is the name of the file .txt.*

- std::string `fileNameBin` = "sensors.dat"

*This is the name of the file .dat.*

- bool `status_` = true

*The status of the alarm.*

### 4.2.1 Detailed Description

Definition at line 25 of file AlarmSensors.h.

### 4.2.2 Constructor & Destructor Documentation

#### 4.2.2.1 AlarmSensors()

```
AlarmSensors::AlarmSensors ( ) [explicit]
```

Construct a new Alarm Sensors object.

Definition at line 19 of file AlarmSensors.cpp.

```
19
20 // El set de sensores se inicializa con un sensor de cada tipo
21 sensors_.insert(new TemperatureSensor(1, true));
22 sensors_.insert(new AirQualitySensor(2, true));
23 sensors_.insert(new HydrometerSensor(3, true));
24 sensors_.insert(new PressureSensor(4, true));
25 sensors_.insert(new LightSensor(5, true));
26 sensors_.insert(new PhSensor(6, true));
27 }
```

#### 4.2.2.2 ~AlarmSensors()

```
AlarmSensors::~AlarmSensors ( )
```

Destroy the Alarm Sensors object.

Definition at line 29 of file AlarmSensors.cpp.

```
29
30 // Destructor que elimina todos los sensores
31 for (auto sensor : sensors_) {
32     delete sensor;
33 }
34 }
```

### 4.2.3 Member Function Documentation

#### 4.2.3.1 addSensor()

```
void AlarmSensors::addSensor (
    int id,
    std::string type )
```

Add a [Sensor](#) object.

##### Parameters

<i>id</i>	
<i>type</i>	

Definition at line 46 of file AlarmSensors.cpp.

```
46
47 // Si el sensor ya existe no se puede añadir
48 if (sensorExists(id)) {
49     cout << "Sensor already exists" << endl;
50     return;
51 }
```

```

52 // Pasar a mayusculas el tipo de sensor
53 for (auto &c : type) {
54     c = toupper(c);
55 }
56 // Añadir un sensor al set de sensores
57 if (type == "TEMPERATURE") {
58     sensors_.insert(new TemperatureSensor(id, true));
59 } else if (type == "AIR_QUALITY") {
60     sensors_.insert(new AirQualitySensor(id, true));
61 } else if (type == "HYDROMETER") {
62     sensors_.insert(new HydrometerSensor(id, true));
63 } else if (type == "PRESSURE") {
64     sensors_.insert(new PressureSensor(id, true));
65 } else if (type == "LIGHT") {
66     sensors_.insert(new LightSensor(id, true));
67 } else if (type == "PH") {
68     sensors_.insert(new PhSensor(id, true));
69 } else {
70     cout << "Sensor type not valid" << endl;
71 }
72 }
```

Referenced by GreenHouse::manageAdmin(), and GreenHouse::manageEmployee().

Here is the caller graph for this function:



#### 4.2.3.2 checkAllgood()

```
bool AlarmSensors::checkAllgood () [private]
```

Check if the Sensors have good measurements.

##### Returns

true if the Sensors have good measurements  
false if the Sensors do not have good measurements

Definition at line 102 of file AlarmSensors.cpp.

```

102 {
103     // Si los sensores tienen buenas mediciones
104     for (auto sensor : sensors_) {
105         if (!sensor->checkAllgood()) {
106             return false;
107         }
108     }
109     return true;
110 }
```

#### 4.2.3.3 checkSensors()

```
int AlarmSensors::checkSensors ( ) [private]
```

Check the Sensors.

##### Returns

int

Definition at line 112 of file AlarmSensors.cpp.

```
112     {
113     if (!sensorsInitialized()) {
114         return -1;
115     } else {
116         if (checkAllgood()) {
117             return 1;
118         } else {
119             return 0;
120         }
121     }
122 }
```

#### 4.2.3.4 deleteSensor()

```
void AlarmSensors::deleteSensor (
    int id )
```

Delete a [Sensor](#) object.

##### Parameters

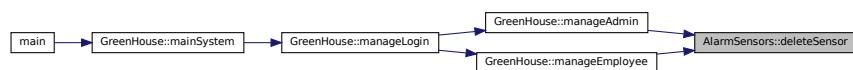
<i>id</i>	
-----------	--

Definition at line 74 of file AlarmSensors.cpp.

```
74     {
75     bool found = false;
76     // Eliminar un sensor del set de sensores
77     for (auto sensor : sensors_) {
78         if (sensor->getId() == id) {
79             std::cout << "Sensor with id: " << id << " deleted" << std::endl;
80             sensors_.erase(sensor);
81             delete sensor;
82             found = true;
83             break;
84         }
85     }
86     if (!found) {
87         std::cout << "Sensor with id: " << id << " not found" << std::endl;
88     }
89 }
```

Referenced by [GreenHouse::manageAdmin\(\)](#), and [GreenHouse::manageEmployee\(\)](#).

Here is the caller graph for this function:



#### 4.2.3.5 displayAlarmStatus()

```
void AlarmSensors::displayAlarmStatus ( )
```

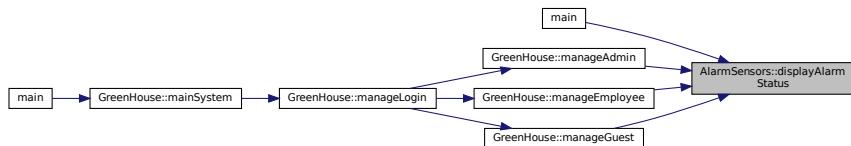
Display the Alarm Status.

Definition at line 124 of file AlarmSensors.cpp.

```
124
125   if (status_) {
126     if (checkSensors() == 1) {
127       cout << "-----" << endl;
128       cout << "|";
129       cout << "|      All sensors are in good status" << endl;
130       cout << "|";
131       cout << "-----" << endl;
132   } else if (checkSensors() == 0) {
133     // Dibujar el logo de alarma, el triangulo con la exclamacion en el medio
134     cout << "-----" << endl;
135     cout << "|";
136     cout << "|  ;One or more sensors are not in good status;" << endl;
137     cout << "|";
138     cout << "-----" << endl;
139   } else if (checkSensors() == -1) {
140     cout << "-----" << endl;
141     cout << "|";
142     cout << "|  ;One or more sensors are not initialized;" << endl;
143     cout << "|    do a collect of data to initialize them" << endl;
144     cout << "|";
145     cout << "-----" << endl;
146   } else {
147     cout << "-----" << endl;
148     cout << "|";
149     cout << "|      ;Error in the system;" << endl;
150     cout << "|";
151     cout << "-----" << endl;
152   }
153 } else {
154   cout << "-----" << endl;
155   cout << "|";
156   cout << "|      The system its off" << endl;
157   cout << "|";
158   cout << "-----" << endl;
159 }
160 }
```

Referenced by main(), GreenHouse::manageAdmin(), GreenHouse::manageEmployee(), and GreenHouse::manageGuest().

Here is the caller graph for this function:



#### 4.2.3.6 `displayAllSensorsData()`

```
void AlarmSensors::displayAllSensorsData ( )
```

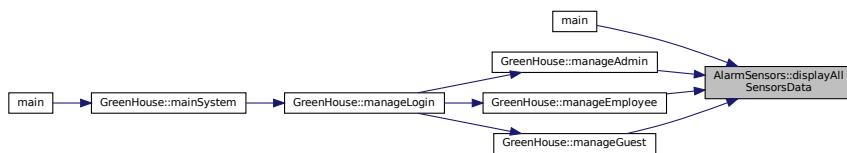
Display all Sensors Data.

Definition at line 162 of file `AlarmSensors.cpp`.

```
162          {
163      // Imprimir los datos de todos los sensores
164      for (auto sensor : sensors_) {
165          sensor->collectAndPrint();
166      }
167 }
```

Referenced by `main()`, `GreenHouse::manageAdmin()`, `GreenHouse::manageEmployee()`, and `GreenHouse::manageGuest()`.

Here is the caller graph for this function:



#### 4.2.3.7 `loadSensorsData()`

```
void AlarmSensors::loadSensorsData ( )
```

This method loads the sensors data from a file `.dat`, but you can change to loads the sensor from a `.txt`.

Definition at line 290 of file `AlarmSensors.cpp`.

```
290          {
291      // Cargar los datos de los sensores de un archivo binario
292      loadSensorsDataBin();
293      // Cargar los datos de los sensores de un archivo de texto
294      // loadSensorsDataTxt();
295 }
```

Referenced by `GreenHouse::manageLogin()`.

Here is the caller graph for this function:



#### 4.2.3.8 loadSensorsDataBin()

```
void AlarmSensors::loadSensorsDataBin ( )
```

This method loads the sensors data from a file .dat.

Definition at line 262 of file AlarmSensors.cpp.

```
262 {
263     // Cargar los datos de los sensores de un archivo binario usando trunc
264     ifstream file(fileNameBin, ios::binary);
265     int id;
266     string type;
267     int data;
268     while (file.read(reinterpret_cast<char *>(&id), sizeof(int))) {
269         // Strings cannot be read directly as binary data due to their dynamic size
270         // First, read the length of the string
271         size_t typeLength;
272         if (file.read(reinterpret_cast<char *>(&typeLength), sizeof(size_t))) {
273             // Resize the string to the read length
274             type.resize(typeLength);
275             // Now, read the string data
276             file.read(&type[0], typeLength);
277         }
278         file.read(reinterpret_cast<char *>(&data), sizeof(int));
279         addSensor(id, type);
280         for (auto sensor : sensors_) {
281             if (sensor->getId() == id) {
282                 sensor->setData(data);
283             }
284         }
285         std::cout << "Sensor with id: " << id << " loaded (binary)" << std::endl;
286     }
287     file.close();
288 }
```

#### 4.2.3.9 loadSensorsDataTxt()

```
void AlarmSensors::loadSensorsDataTxt ( )
```

This method loads the sensors data from a file .txt.

Definition at line 243 of file AlarmSensors.cpp.

```
243 {
244     // Cargar los datos de los sensores de un archivo de texto
245     ifstream file;
246     file.open(fileNameTxt);
247     int id;
248     string type;
249     int data;
250     while (file >> id >> type >> data) {
251         addSensor(id, type);
252         for (auto sensor : sensors_) {
253             if (sensor->getId() == id) {
254                 sensor->setData(data);
255             }
256         }
257         std::cout << "Sensor with id: " << id << " loaded (txt)" << std::endl;
258     }
259     file.close();
260 }
```

#### 4.2.3.10 saveSensorsData()

```
void AlarmSensors::saveSensorsData ( )
```

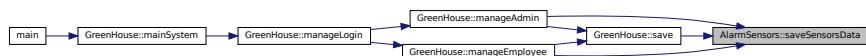
This method saves the sensors data to a file, one .txt and other one .dat.

Definition at line 236 of file AlarmSensors.cpp.

```
236      {
237      // Guardar los datos de los sensores en un archivo binario
238      saveSensorsDataBin();
239      // GUardar los datos de los sensores en un archivo de texto
240      saveSensorsDataTxt();
241  }
```

Referenced by GreenHouse::manageAdmin(), GreenHouse::manageEmployee(), and GreenHouse::save().

Here is the caller graph for this function:



#### 4.2.3.11 saveSensorsDataBin()

```
void AlarmSensors::saveSensorsDataBin ( )
```

This method saves the sensors data to a file .dat.

Definition at line 217 of file AlarmSensors.cpp.

```
217      {
218      ofstream file(fileNameBin, ios::binary | ios::trunc);
219      for (auto sensor : sensors_) {
220          int id = sensor->getId();
221          file.write(reinterpret_cast<char *>(&id), sizeof(int));
222
223          size_t typeLength = sensor->getType().length();
224          file.write(reinterpret_cast<char *>(&typeLength), sizeof(size_t));
225          file.write(sensor->getType().c_str(), typeLength);
226
227          int data = sensor->getData();
228          file.write(reinterpret_cast<char *>(&data), sizeof(int));
229
230          std::cout << "Sensor with id: " << sensor->getId() << " saved (binary)"
231              << std::endl;
232      }
233      file.close();
234  }
```

#### 4.2.3.12 saveSensorsDataTxt()

```
void AlarmSensors::saveSensorsDataTxt ( )
```

This method saves the sensors data to a file .txt.

Definition at line 204 of file AlarmSensors.cpp.

```
204      {
205      // Guardar los datos de los sensores en un archivo de texto
206      ofstream file;
207      file.open(fileNameTxt);
208      for (auto sensor : sensors_) {
209          file << sensor->getId() << " " << sensor->getType() << " "
210              << sensor->getData() << endl;
211          std::cout << "Sensor with id: " << sensor->getId() << " saved (txt)"
212              << std::endl;
213      }
214      file.close();
215  }
```

#### 4.2.3.13 sensorExists()

```
bool AlarmSensors::sensorExists (
    int id) [private]
```

Check if a [Sensor](#) exists.

##### Parameters

<i>id</i>	
-----------	--

##### Returns

true if the [Sensor](#) exists  
false if the [Sensor](#) does not exist

Definition at line 36 of file AlarmSensors.cpp.

```
36
37 // Ver si existe un sensor
38 for (auto sensor : sensors_) {
39     if (sensor->getId() == id) {
40         return true;
41     }
42 }
43 return false;
44 }
```

#### 4.2.3.14 sensorsInitialized()

```
bool AlarmSensors::sensorsInitialized () [private]
```

Check if the Sensors are Initialized.

##### Returns

true if the Sensors are Initialized  
false if the Sensors are not Initialized

Definition at line 91 of file AlarmSensors.cpp.

```
91
92 // Los sensores estan iniciados si todos los sensores no tienen el valor por
93 // defecto de -1
94 for (auto sensor : sensors_) {
95     if (sensor->getData() == -1) {
96         return false;
97     }
98 }
99 return true;
100 }
```

#### 4.2.3.15 turnOffSystem()

```
void AlarmSensors::turnOffSystem ( ) [private]
```

Turn off the System.

Definition at line 176 of file AlarmSensors.cpp.

```
176     {
177     // Apagar todos los sensores del set
178     for (auto sensor : sensors_) {
179         sensor->turnOff();
180     }
181 }
```

#### 4.2.3.16 turnOnOffSystem()

```
void AlarmSensors::turnOnOffSystem (
    int input )
```

This method turns on or off the system.

##### Parameters

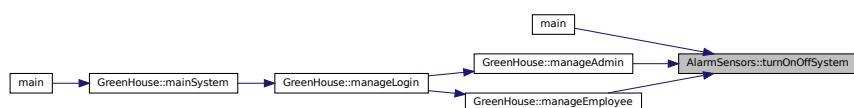
<i>input</i>	<input type="text"/>
--------------	----------------------

Definition at line 183 of file AlarmSensors.cpp.

```
183     {
184     // Si el input es igual a true entonces encender todos los sensores
185     if (input == 1) {
186         turnOnSystem();
187         cout << "-----" << endl;
188         cout << "| " << endl;
189         cout << "| " << "System turned on" << endl;
190         cout << "| " << endl;
191         cout << "-----" << endl;
192         status_ = true;
193     } else if (input == 2) {
194         turnOffSystem();
195         cout << "-----" << endl;
196         cout << "| " << endl;
197         cout << "| " << "System turned off" << endl;
198         cout << "| " << endl;
199         cout << "-----" << endl;
200         status_ = false;
201     }
202 }
```

Referenced by main(), GreenHouse::manageAdmin(), and GreenHouse::manageEmployee().

Here is the caller graph for this function:



#### 4.2.3.17 turnOnSystem()

```
void AlarmSensors::turnOnSystem ( ) [private]
```

Turn on the System.

Definition at line 169 of file AlarmSensors.cpp.

```
169
170   // Encender todos los sensores del set
171   for (auto sensor : sensors_) {
172     sensor->turnOn();
173   }
174 }
```

### 4.2.4 Member Data Documentation

#### 4.2.4.1 fileNameBin

```
std::string AlarmSensors::fileNameBin = "sensors.dat" [private]
```

This is the name of the file .dat.

Definition at line 131 of file AlarmSensors.h.

#### 4.2.4.2 fileNameTxt

```
std::string AlarmSensors::fileNameTxt = "sensors.txt" [private]
```

This is the name of the file .txt.

Definition at line 126 of file AlarmSensors.h.

#### 4.2.4.3 sensors\_

```
std::set<Sensor *> AlarmSensors::sensors_ [private]
```

This is the set of [Sensor](#) pointers.

Definition at line 119 of file AlarmSensors.h.

#### 4.2.4.4 `status_`

```
bool AlarmSensors::status_ = true [private]
```

The status of the alarm.

Definition at line 183 of file `AlarmSensors.h`.

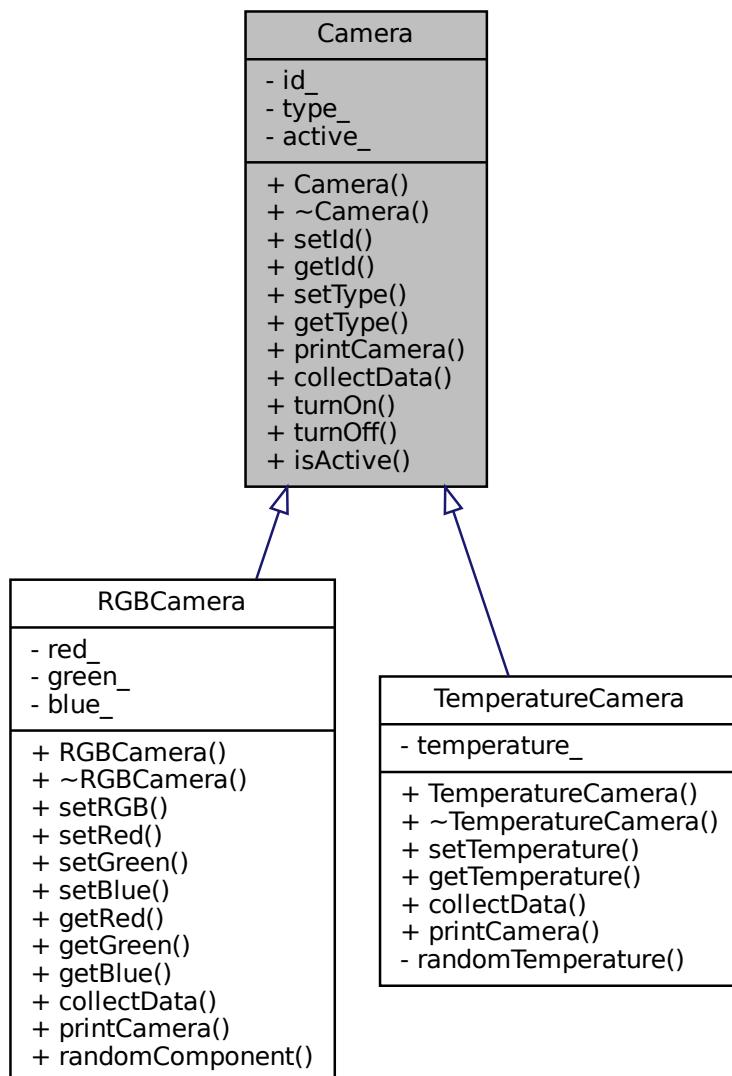
The documentation for this class was generated from the following files:

- src/[AlarmSensors.h](#)
- src/[AlarmSensors.cpp](#)

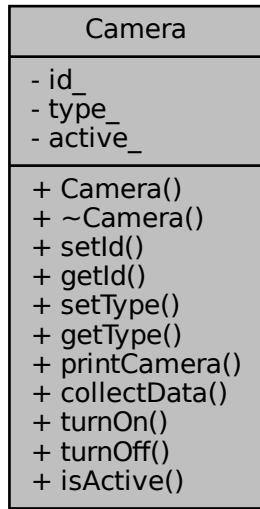
## 4.3 Camera Class Reference

```
#include <Camera.h>
```

Inheritance diagram for Camera:



Collaboration diagram for Camera:



## Public Member Functions

- **Camera** (int id, std::string type, bool active)  
*Construct a new Camera object.*
- virtual **~Camera** ()  
*Destroy the Camera object.*
- void **setId** (int newid)  
*Set the id of the camera.*
- int **getId** () const  
*Get the id of the camera.*
- void **setType** (std::string newtype)  
*Set the type of the camera.*
- std::string **getType** () const  
*Get the type of the camera.*
- virtual void **printCamera** ()  
*Print the camera information.*
- virtual void **collectData** ()  
*Collect data from the camera.*
- void **turnOn** ()  
*Turn on the camera.*
- void **turnOff** ()  
*Turn off the camera.*
- const bool **isActive** ()  
*Check if the camera is active.*

## Private Attributes

- int `id_`  
*This attribute represents the id of the camera.*
- std::string `type_`  
*This attribute represents the type of the camera.*
- bool `active_`  
*This attribute represents if the camera is active.*

### 4.3.1 Detailed Description

Definition at line 16 of file Camera.h.

### 4.3.2 Constructor & Destructor Documentation

#### 4.3.2.1 Camera()

```
Camera::Camera (
    int id,
    std::string type,
    bool active ) [explicit]
```

Construct a new [Camera](#) object.

#### Parameters

<code>id</code>	
<code>type</code>	
<code>active</code>	

#### Returns

[Camera](#) object

Definition at line 5 of file Camera.cpp.

```
5
6     id_ = id;
7     type_ = type;
8     active_ = active;
9 }
```

#### 4.3.2.2 ~Camera()

```
Camera::~Camera ( ) [virtual]
```

Destroy the [Camera](#) object.

Definition at line 11 of file Camera.cpp.

```
11 { }
```

### 4.3.3 Member Function Documentation

#### 4.3.3.1 collectData()

```
void Camera::collectData ( ) [virtual]
```

Collect data from the camera.

Reimplemented in [TemperatureCamera](#), and [RGBCamera](#).

Definition at line 21 of file Camera.cpp.

```
21           {
22     cout << "Collecting data from camera " << id_ << endl;
23 }
```

#### 4.3.3.2 getId()

```
int Camera::getId ( ) const
```

Get the id of the camera.

Returns

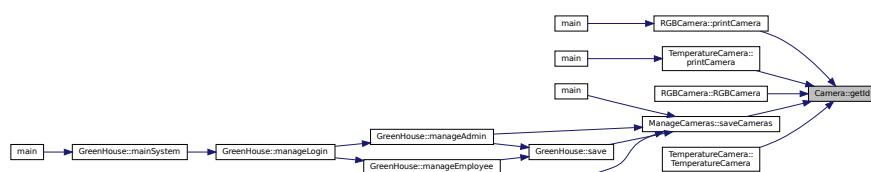
int

Definition at line 15 of file Camera.cpp.

```
15 { return id_; }
```

Referenced by [RGBCamera::printCamera\(\)](#), [TemperatureCamera::printCamera\(\)](#), [RGBCamera::RGBCamera\(\)](#), [ManageCameras::saveCameras\(\)](#), and [TemperatureCamera::TemperatureCamera\(\)](#).

Here is the caller graph for this function:



### 4.3.3.3 getType()

```
std::string Camera::getType ( ) const
```

Get the type of the camera.

Returns

```
std::string
```

Definition at line 19 of file Camera.cpp.

```
19 { return type_; }
```

### 4.3.3.4 isActive()

```
const bool Camera::isActive ( )
```

Check if the camera is active.

Returns

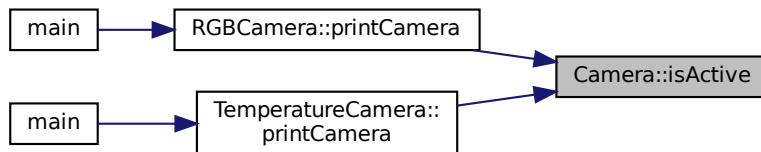
```
const bool
```

Definition at line 34 of file Camera.cpp.

```
34 { return active_; }
```

Referenced by RGBCamera::printCamera(), and TemperatureCamera::printCamera().

Here is the caller graph for this function:



### 4.3.3.5 printCamera()

```
void Camera::printCamera ( ) [virtual]
```

Print the camera information.

Reimplemented in [TemperatureCamera](#), and [RGBCamera](#).

Definition at line 25 of file Camera.cpp.

```
25 {
26   cout << "Camera id: " << id_ << " type: " << type_ << " active: " << active_
27   << endl;
28 }
```

#### 4.3.3.6 setId()

```
void Camera::setId (
    int newid )
```

Set the id of the camera.

**Parameters**

<i>newid</i>	<input type="button" value=""/>
--------------	---------------------------------

Definition at line 13 of file Camera.cpp.

```
13 { id_ = newid; }
```

**4.3.3.7 setType()**

```
void Camera::setType (  
    std::string newtype )
```

Set the type of the camera.

**Parameters**

<i>newtype</i>	<input type="button" value=""/>
----------------	---------------------------------

Definition at line 17 of file Camera.cpp.

```
17 { type_ = newtype; }
```

**4.3.3.8 turnOff()**

```
void Camera::turnOff ( )
```

Turn off the camera.

Definition at line 32 of file Camera.cpp.

```
32 { active_ = false; }
```

Referenced by main().

Here is the caller graph for this function:



#### 4.3.3.9 turnOn()

```
void Camera::turnOn ( )
```

Turn on the camera.

Definition at line 30 of file Camera.cpp.

```
30 { active_ = true; }
```

### 4.3.4 Member Data Documentation

#### 4.3.4.1 active\_

```
bool Camera::active_ [private]
```

This attribute represents if the camera is active.

Definition at line 98 of file Camera.h.

#### 4.3.4.2 id\_

```
int Camera::id_ [private]
```

This attribute represents the id of the camera.

Definition at line 88 of file Camera.h.

#### 4.3.4.3 type\_

```
std::string Camera::type_ [private]
```

This attribute represents the type of the camera.

Definition at line 93 of file Camera.h.

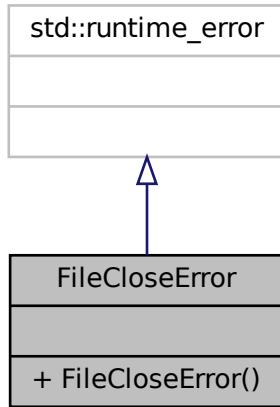
The documentation for this class was generated from the following files:

- src/[Camera.h](#)
- src/[Camera.cpp](#)

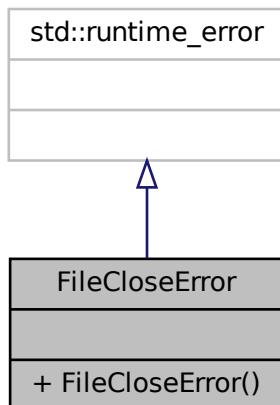
## 4.4 FileCloseError Class Reference

```
#include <Exceptions.h>
```

Inheritance diagram for FileCloseError:



Collaboration diagram for FileCloseError:



### Public Member Functions

- `FileCloseError` (const std::string &filename)

*Construct a new File Close Error object.*

#### 4.4.1 Detailed Description

Definition at line 25 of file Exceptions.h.

#### 4.4.2 Constructor & Destructor Documentation

##### 4.4.2.1 FileCloseError()

```
FileCloseError::FileCloseError (
    const std::string & filename ) [inline], [explicit]
```

Construct a new File Close Error object.

###### Parameters

<i>filename</i>	
-----------------	--

Definition at line 32 of file Exceptions.h.

```
33     : std::runtime_error("Error closing file: " + filename) {}
```

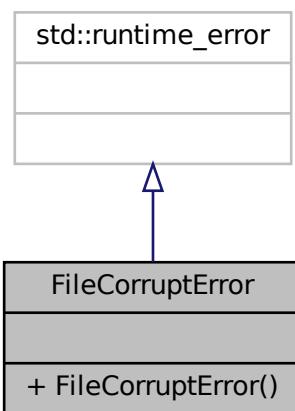
The documentation for this class was generated from the following file:

- src/[Exceptions.h](#)

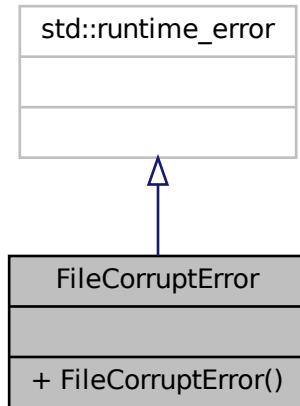
## 4.5 FileCorruptError Class Reference

```
#include <Exceptions.h>
```

Inheritance diagram for FileCorruptError:



Collaboration diagram for FileCorruptError:



## Public Member Functions

- [FileCorruptError](#) (const std::string &filename)

*Construct a new File Corrupt Error object.*

### 4.5.1 Detailed Description

Definition at line 91 of file Exceptions.h.

### 4.5.2 Constructor & Destructor Documentation

#### 4.5.2.1 FileCorruptError()

```
FileCorruptError::FileCorruptError (
    const std::string & filename ) [inline], [explicit]
```

Construct a new File Corrupt Error object.

##### Parameters

<i>filename</i>	<input type="text"/>
-----------------	----------------------

Definition at line 98 of file Exceptions.h.

```
99         : std::runtime_error("File is corrupt: " + filename) {}
```

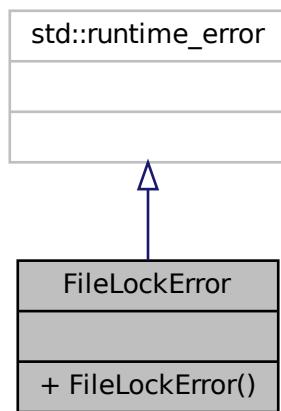
The documentation for this class was generated from the following file:

- src/[Exceptions.h](#)

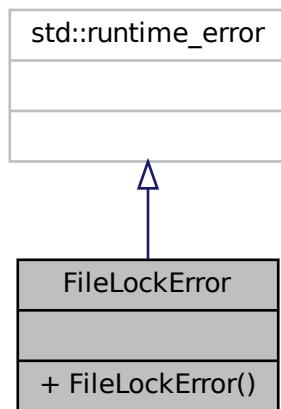
## 4.6 FileLockError Class Reference

```
#include <Exceptions.h>
```

Inheritance diagram for FileLockError:



Collaboration diagram for FileLockError:



## Public Member Functions

- [FileLockError](#) (const std::string &filename)

*Construct a new File Lock Error object.*

### 4.6.1 Detailed Description

Definition at line 80 of file Exceptions.h.

### 4.6.2 Constructor & Destructor Documentation

#### 4.6.2.1 FileLockError()

```
FileLockError::FileLockError (
    const std::string & filename ) [inline], [explicit]
```

Construct a new File Lock Error object.

##### Parameters

<i>filename</i>	<input type="text"/>
-----------------	----------------------

Definition at line 87 of file Exceptions.h.

```
88     : std::runtime_error("File is locked: " + filename) {}
```

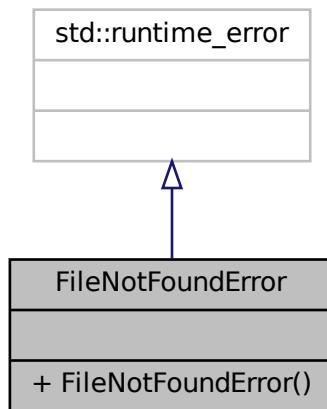
The documentation for this class was generated from the following file:

- src/[Exceptions.h](#)

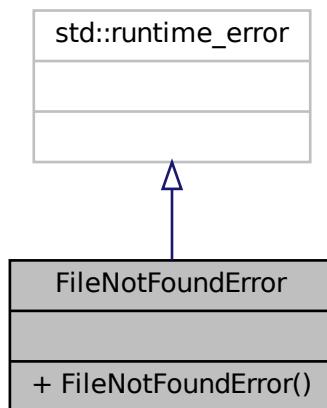
## 4.7 FileNotFoundException Class Reference

```
#include <Exceptions.h>
```

Inheritance diagram for FileNotFoundException:



Collaboration diagram for FileNotFoundException:



## Public Member Functions

- [FileNotFoundException](#) (const std::string &filename)  
*Construct a new File Not Found Error object.*

### 4.7.1 Detailed Description

Definition at line 69 of file Exceptions.h.

## 4.7.2 Constructor & Destructor Documentation

### 4.7.2.1 FileNotFoundException()

```
FileNotFoundException::FileNotFoundException (
    const std::string & filename ) [inline], [explicit]
```

Construct a new File Not Found Error object.

#### Parameters

<i>filename</i>	
-----------------	--

Definition at line 76 of file Exceptions.h.

```
77     : std::runtime_error("File not found: " + filename) {}
```

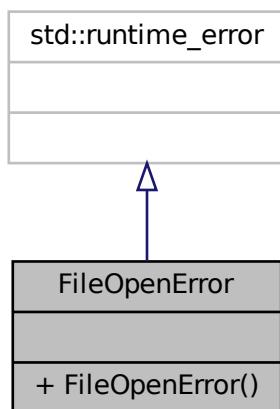
The documentation for this class was generated from the following file:

- src/[Exceptions.h](#)

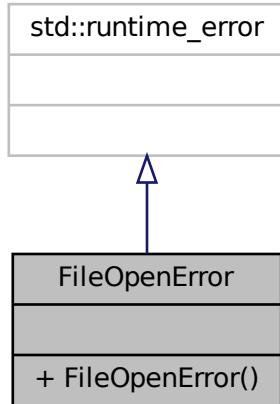
## 4.8 FileOpenError Class Reference

```
#include <Exceptions.h>
```

Inheritance diagram for FileOpenError:



Collaboration diagram for FileOpenError:



## Public Member Functions

- [FileOpenError](#) (const std::string &filename)

*Construct a new File Open Error object.*

### 4.8.1 Detailed Description

Definition at line 14 of file Exceptions.h.

### 4.8.2 Constructor & Destructor Documentation

#### 4.8.2.1 FileOpenError()

```
FileOpenError::FileOpenError (
    const std::string & filename ) [inline], [explicit]
```

Construct a new File Open Error object.

##### Parameters

<code>filename</code>	<input type="text"/>
-----------------------	----------------------

Definition at line 21 of file Exceptions.h.

```
22     : std::runtime_error("Error opening file: " + filename) {}
```

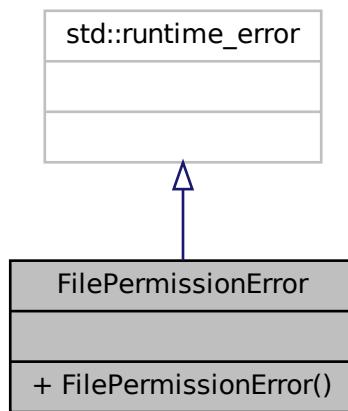
The documentation for this class was generated from the following file:

- src/[Exceptions.h](#)

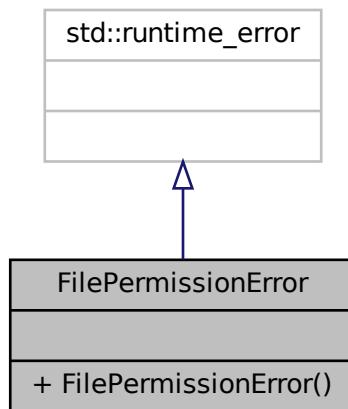
## 4.9 FilePermissionError Class Reference

```
#include <Exceptions.h>
```

Inheritance diagram for FilePermissionError:



Collaboration diagram for FilePermissionError:



## Public Member Functions

- [FilePermissionError](#) (const std::string &filename)

*Construct a new File Permission Error object.*

### 4.9.1 Detailed Description

Definition at line 58 of file Exceptions.h.

### 4.9.2 Constructor & Destructor Documentation

#### 4.9.2.1 FilePermissionError()

```
FilePermissionError::FilePermissionError (const std::string & filename) [inline], [explicit]
```

Construct a new File Permission Error object.

##### Parameters

<i>filename</i>	<input type="text"/>
-----------------	----------------------

Definition at line 65 of file Exceptions.h.

```
66 : std::runtime_error("Permission denied: " + filename) {}
```

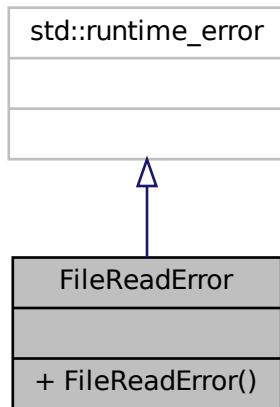
The documentation for this class was generated from the following file:

- src/[Exceptions.h](#)

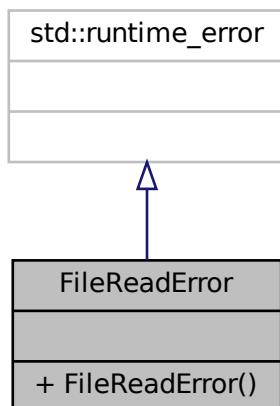
## 4.10 FileReadError Class Reference

```
#include <Exceptions.h>
```

Inheritance diagram for FileReadError:



Collaboration diagram for FileReadError:



## Public Member Functions

- [FileReadError](#) (const std::string &filename)  
*Construct a new File Read Error object.*

### 4.10.1 Detailed Description

Definition at line 36 of file Exceptions.h.

## 4.10.2 Constructor & Destructor Documentation

### 4.10.2.1 FileReadError()

```
FileReadError::FileReadError (
    const std::string & filename ) [inline], [explicit]
```

Construct a new File Read Error object.

#### Parameters

<i>filename</i>	
-----------------	--

Definition at line 43 of file Exceptions.h.

```
44     : std::runtime_error("Error reading file: " + filename) {}
```

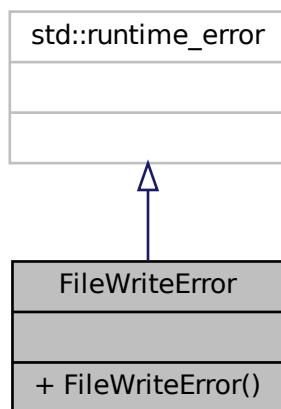
The documentation for this class was generated from the following file:

- src/[Exceptions.h](#)

## 4.11 FileWriteError Class Reference

```
#include <Exceptions.h>
```

Inheritance diagram for FileWriteError:



Collaboration diagram for `FileWriteError`:



## Public Member Functions

- [FileWriteError](#) (const std::string &filename)

*Construct a new File Write Error object.*

### 4.11.1 Detailed Description

Definition at line 47 of file Exceptions.h.

### 4.11.2 Constructor & Destructor Documentation

#### 4.11.2.1 FileWriteError()

```
FileWriteError::FileWriteError (
    const std::string & filename ) [inline], [explicit]
```

Construct a new File Write Error object.

##### Parameters

<i>filename</i>	
-----------------	--

Definition at line 54 of file Exceptions.h.

```
55     : std::runtime_error("Error writing file: " + filename) {}
```

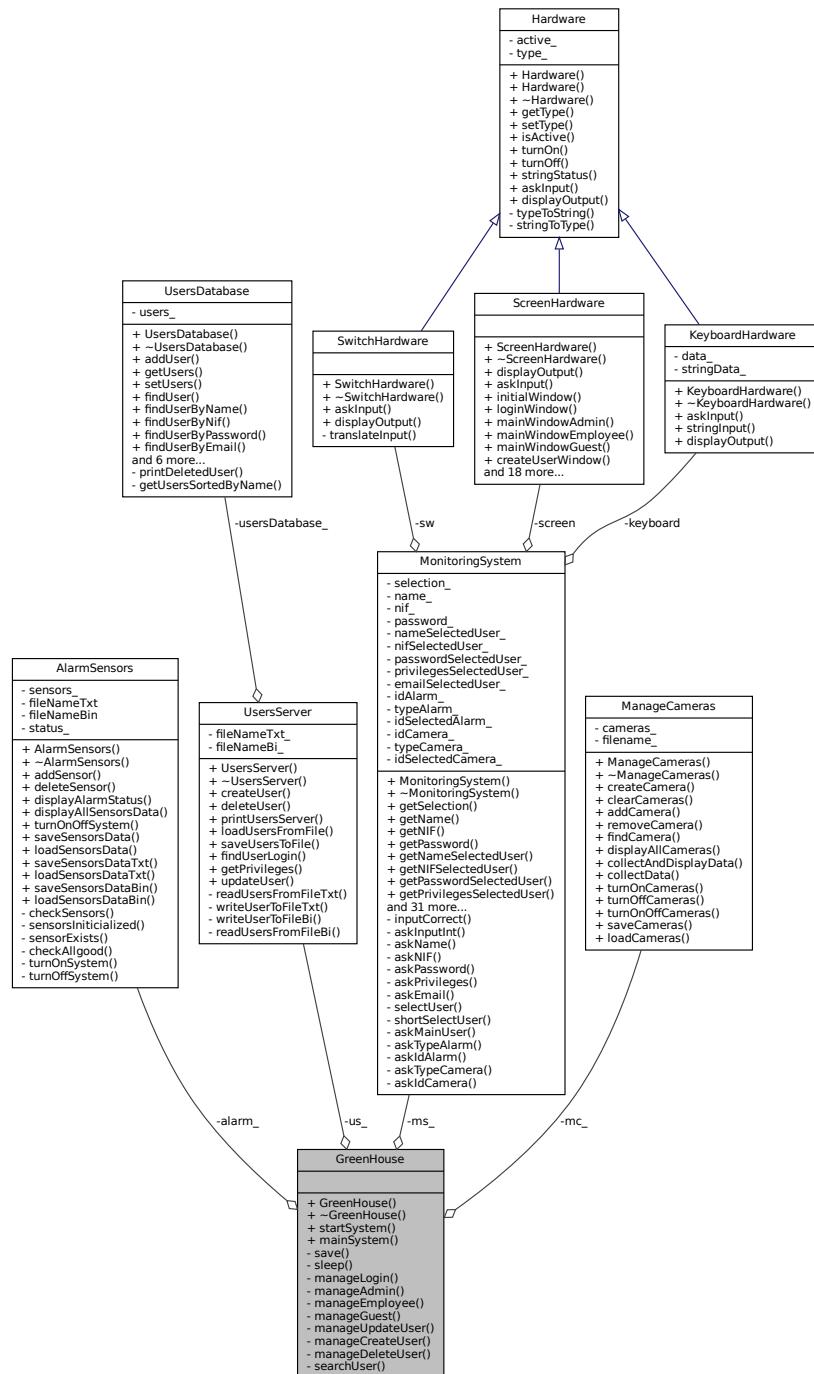
The documentation for this class was generated from the following file:

- src/Exceptions.h

## 4.12 GreenHouse Class Reference

```
#include <GreenHouse.h>
```

Collaboration diagram for GreenHouse:



## Public Member Functions

- `GreenHouse ()`  
*Construct a new Green House object.*
- `~GreenHouse ()`  
*Destroy the Green House object.*
- `void startSystem ()`  
*Start the system.*
- `void mainSystem ()`  
*Main system, after the system is started, you call this method to manage the system with your options that contains your privilege.*

## Private Member Functions

- `void save (int id)`  
*Save the system.*
- `void sleep ()`  
*Sleep the system.*
- `void manageLogin ()`  
*Manage the login.*
- `void manageAdmin ()`  
*Manage the admin.*
- `void manageEmployee ()`  
*Manage the employee.*
- `void manageGuest ()`  
*Manage the guest.*
- `void manageUpdateUser ()`  
*Manage the update user.*
- `void manageCreateUser ()`  
*Manage the create user.*
- `void manageDeleteUser ()`  
*Manage the delete user.*
- `bool searchUser (std::string name, std::string password, std::string nif)`  
*Manage search user.*

## Private Attributes

- `AlarmSensors * alarm_`  
*This attribute is the `AlarmSensors` object (set of pointers to sensors).*
- `MonitoringSystem * ms_`  
*This attribute is the `MonitoringSystem` object.*
- `UsersServer * us_`  
*This attribute is the `UsersServer` object (set of pointers to users).*
- `ManageCameras * mc_`  
*This attribute is the `ManageCameras` object (set of pointers to cameras).*

### 4.12.1 Detailed Description

Definition at line 17 of file GreenHouse.h.

## 4.12.2 Constructor & Destructor Documentation

### 4.12.2.1 GreenHouse()

```
GreenHouse::GreenHouse ( )
```

Construct a new Green House object.

Definition at line 17 of file GreenHouse.cpp.

```
18     : alarm_(new AlarmSensors()),
19
20     ms_(new MonitoringSystem(new ScreenHardware(true),
21                             new KeyboardHardware(true),
22                             new SwitchHardware(true))),
23     us_(new UsersServer()),
24
25     mc_(new ManageCameras()) {
26     // Constructor ahora inicializa todos los atributos privados correctamente.
27 }
```

### 4.12.2.2 ~GreenHouse()

```
GreenHouse::~GreenHouse ( )
```

Destroy the Green House object.

Definition at line 29 of file GreenHouse.cpp.

```
29     {
30     // Destructor ahora elimina todos los atributos privados correctamente.
31     delete alarm_;
32     delete ms_;
33     delete us_;
34     delete mc_;
35 }
```

References alarm\_, mc\_, ms\_, and us\_.

## 4.12.3 Member Function Documentation

#### 4.12.3.1 mainSystem()

```
void GreenHouse::mainSystem ( )
```

Main system, after the system is started, you call this method to manage the system with your options that contains your privilege.

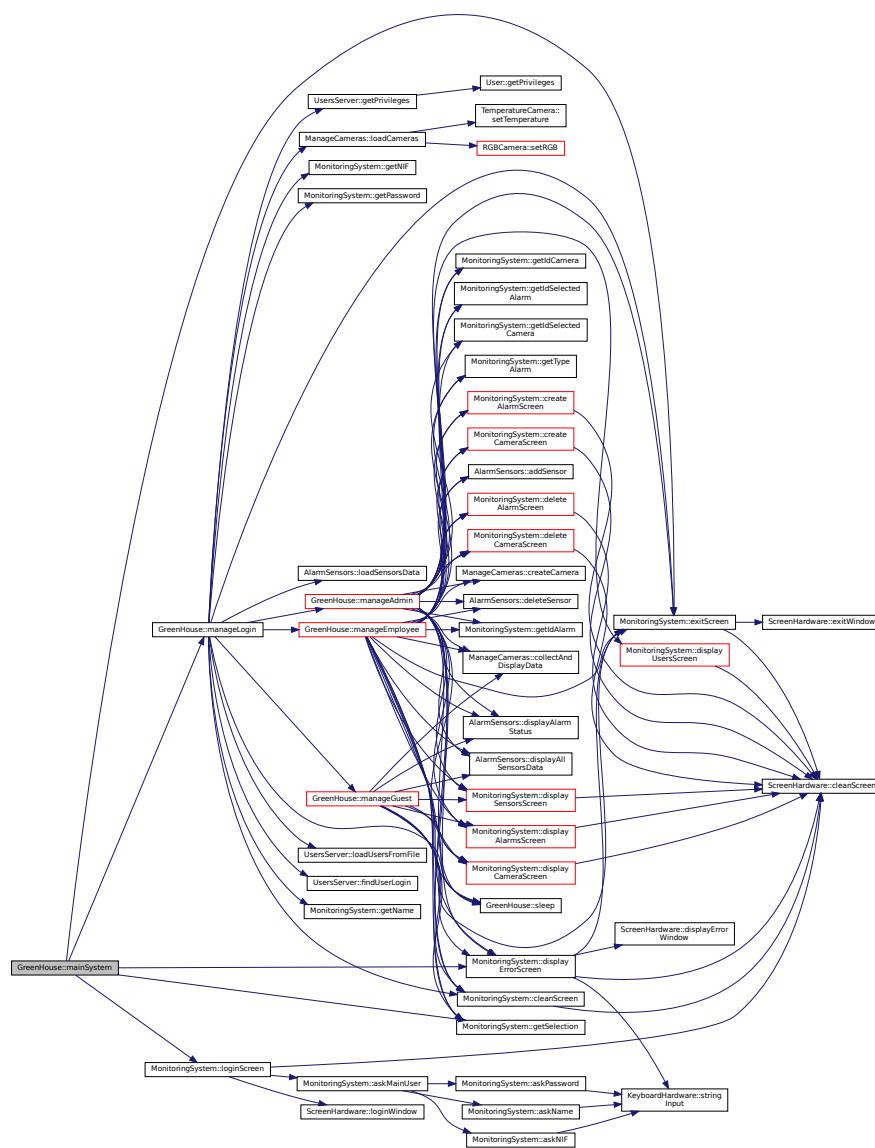
Definition at line 317 of file GreenHouse.cpp.

```
317     {
318     // SI la seleccion en startSystem() es 1, entonces se ejecuta el loginScreen()
319     // Hacemos mejor un switch para que sea más fácil de leer
320     switch (ms_->getSelection()) {
321     case 1:
322         ms_->loginScreen();
323         manageLogin();
324         break;
325     case 2:
326         ms_->exitScreen();
327         break;
328     default:
329         ms_->displayErrorScreen();
330         break;
331     }
332     /* if (ms_->getSelection() == 1)
333     {
334         us_->loadUsersFromFile();
335         ms_->loginScreen();
336     } else if (ms_->getSelection() == 2)
337     {
338         ms_->exitScreen();
339     } else {
340         ms_->displayErrorScreen();
341     }
342     */
343 }
```

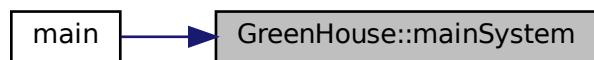
References MonitoringSystem::displayErrorScreen(), MonitoringSystem::exitScreen(), MonitoringSystem::getSelection(), MonitoringSystem::loginScreen(), manageLogin(), and ms\_-.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.12.3.2 manageAdmin()

```
void GreenHouse::manageAdmin ( ) [private]
```

Manage the admin.

Definition at line 91 of file GreenHouse.cpp.

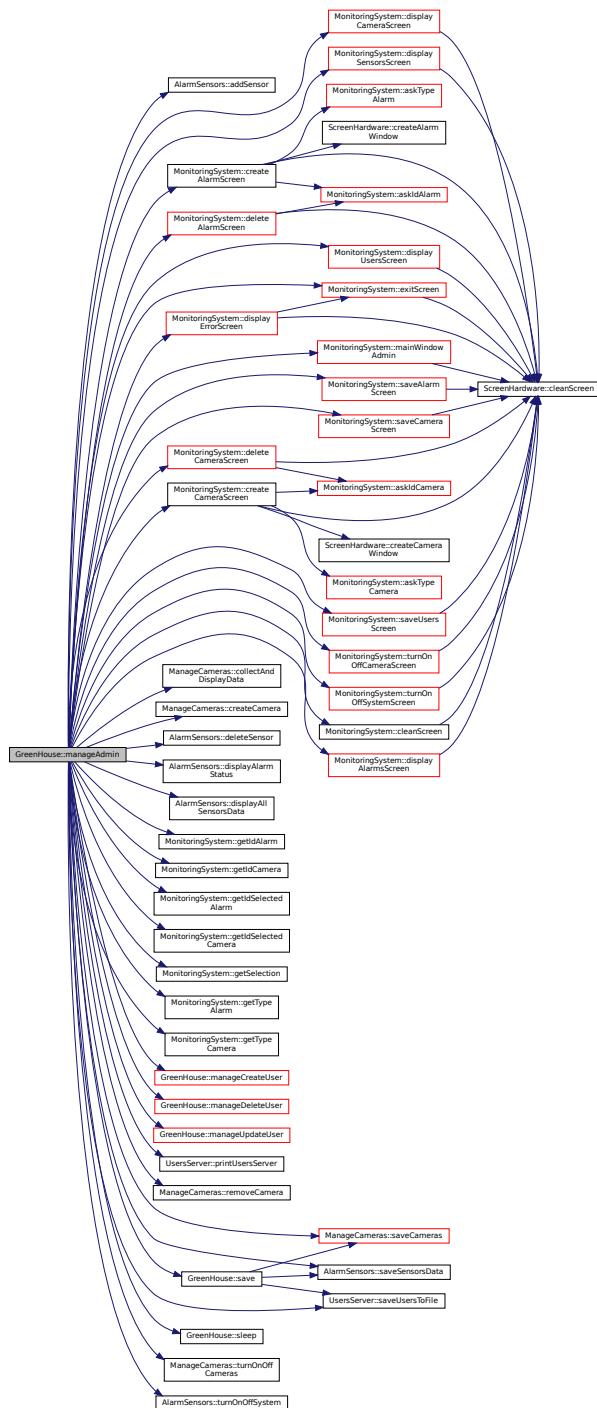
```
91
92     bool exit = false;
93     // Mostramos la ventana de admin
94     do {
95         // Poner unos segundos de espera para que se vea el mensaje
96         sleep();
97         ms_>mainWindowAdmin();
98         switch (ms_>getSelection()) {
99             case 1:
100                 manageCreateUser();
101                 // us_->saveUsersToFile();
102                 break;
103             case 2:
104                 manageDeleteUser();
105                 // us_->saveUsersToFile();
106                 break;
107             case 3:
108                 manageUpdateUser();
109                 // us_->saveUsersToFile();
110                 break;
111             case 4:
112                 ms_->displayUsersScreen();
113                 us_->printUsersServer();
114                 break;
115             case 5:
116                 ms_->createAlarmScreen();
117                 alarm_->addSensor(ms_->getIdAlarm(), ms_->getTypeAlarm());
118                 break;
119             case 6:
120                 ms_->deleteAlarmScreen();
121                 alarm_->deleteSensor(ms_->getIdSelectedAlarm());
122                 break;
123             case 7:
124                 ms_->displaySensorsScreen();
125                 alarm_->displayAllSensorsData();
126                 break;
127             case 8:
128                 ms_->displayAlarmsScreen();
129                 alarm_->displayAlarmStatus();
130                 break;
131             case 9:
132                 ms_->turnOnOffSystemScreen();
133                 alarm_->turnOnOffSystem(ms_->getSelection());
134                 break;
135             case 10:
136                 // Enseñar camaras
137                 ms_->displayCameraScreen();
138                 mc_->collectAndDisplayData();
139                 break;
140             case 11:
141                 // Crear camera
142                 ms_->createCameraScreen();
143                 mc_->createCamera(ms_->getIdCamera(), ms_->getTypeCamera());
144                 break;
145             case 12:
146                 // Borrar camera
147                 ms_->deleteCameraScreen();
148                 mc_->removeCamera(ms_->getIdSelectedCamera());
149                 break;
150             case 13:
151                 // Apagar o encender camaras
152                 ms_->turnOnOffCameraScreen();
153                 mc_->turnOnOffCameras(ms_->getSelection());
154                 break;
155             case 14:
156                 // Guardar camaras
157                 ms_->saveCameraScreen();
158                 mc_->saveCameras();
159                 break;
160             case 15:
161                 ms_->saveUsersScreen();
162                 us_->saveUsersToFile();
163                 break;
164             case 16:
165                 ms_->saveAlarmScreen();
166                 alarm_->saveSensorsData();
```

```
167     break;
168 case 17:
169     ms_>cleanScreen();
170     save(1);
171     sleep();
172     ms_>exitScreen();
173     exit = true;
174     break;
175 default:
176     ms_>displayErrorScreen();
177     break;
178 }
179 } while (!exit);
180 }
```

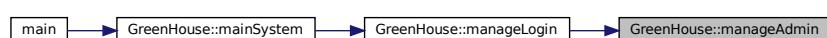
References AlarmSensors::addSensor(), alarm\_, MonitoringSystem::cleanScreen(), ManageCameras::collectAndDisplayData(), MonitoringSystem::createAlarmScreen(), ManageCameras::createCamera(), MonitoringSystem::createCameraScreen(), MonitoringSystem::deleteAlarmScreen(), MonitoringSystem::deleteCameraScreen(), AlarmSensors::deleteSensor(), MonitoringSystem::displayAlarmsScreen(), AlarmSensors::displayAlarmStatus(), AlarmSensors::displayAllSensorsData(), MonitoringSystem::displayCameraScreen(), MonitoringSystem::displayErrorScreen(), MonitoringSystem::displaySensorsScreen(), MonitoringSystem::displayUsersScreen(), MonitoringSystem::exitScreen(), MonitoringSystem::getIdAlarm(), MonitoringSystem::getIdCamera(), MonitoringSystem::getIdSelectedAlarm(), MonitoringSystem::getIdSelectedCamera(), MonitoringSystem::getSelection(), MonitoringSystem::getTypeAlarm(), MonitoringSystem::getTypeCamera(), MonitoringSystem::mainWindowAdmin(), manageCreateUser(), manageDeleteUser(), manageUpdateUser(), mc\_, ms\_, UsersServer::printUsersServer(), ManageCameras::removeCamera(), save(), MonitoringSystem::saveAlarmScreen(), ManageCameras::saveCameras(), MonitoringSystem::saveCameraScreen(), AlarmSensors::saveSensorsData(), MonitoringSystem::saveUsersScreen(), UsersServer::saveUsersToFile(), sleep(), ManageCameras::turnOnOffCameras(), MonitoringSystem::turnOnOffCameraScreen(), AlarmSensors::turnOnOffSystem(), MonitoringSystem::turnOnOffSystemScreen(), and us\_.

Referenced by manageLogin().

Here is the call graph for this function:



Here is the caller graph for this function:



### 4.12.3.3 manageCreateUser()

```
void GreenHouse::manageCreateUser ( ) [private]
```

Manage the create user.

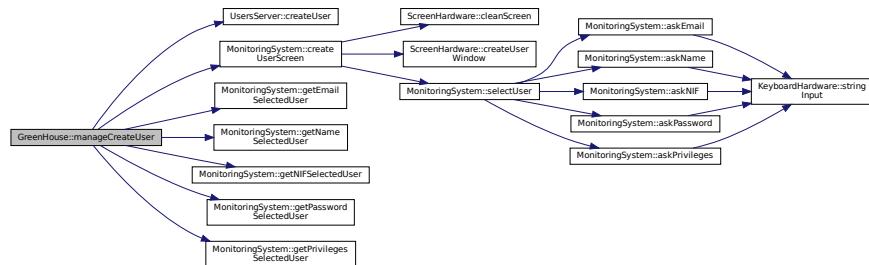
Definition at line 57 of file GreenHouse.cpp.

```
57
58     ms_->createUserScreen();
59     us_->createUser(ms_->getNameSelectedUser(), ms_->getNIFSelectedUser(),
60                       ms_->getPasswordSelectedUser(),
61                       ms_->getPrivilegesSelectedUser(),
62                       ms_->getEmailSelectedUser());
63 }
```

References UsersServer::createUser(), MonitoringSystem::createUserScreen(), MonitoringSystem::getEmailSelectedUser(), MonitoringSystem::getNameSelectedUser(), MonitoringSystem::getNIFSelectedUser(), MonitoringSystem::getPasswordSelectedUser(), MonitoringSystem::getPrivilegesSelectedUser(), ms\_-, and us\_-.

Referenced by manageAdmin().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.12.3.4 manageDeleteUser()

```
void GreenHouse::manageDeleteUser ( ) [private]
```

Manage the delete user.

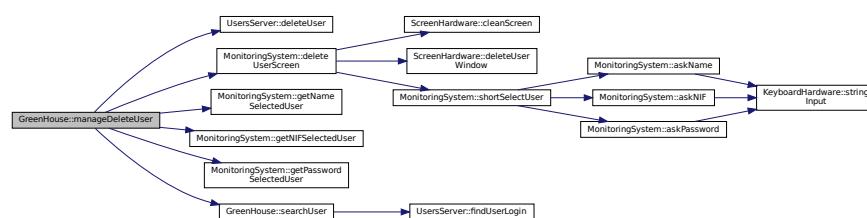
Definition at line 47 of file GreenHouse.cpp.

```
47
48     ms_->deleteUserScreen();
49     if (searchUser(ms_->getNameSelectedUser(), ms_->getPasswordSelectedUser(),
50                  ms_->getNIFSelectedUser()) {
51         us_->deleteUser(ms_->getNIFSelectedUser());
52     } else {
53         printf("Usuario no encontrado\n");
54     }
55 }
```

References UsersServer::deleteUser(), MonitoringSystem::deleteUserScreen(), MonitoringSystem::getNameSelectedUser(), MonitoringSystem::getNIFSelectedUser(), MonitoringSystem::getPasswordSelectedUser(), ms\_, searchUser(), and us\_.

Referenced by manageAdmin().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.12.3.5 manageEmployee()

```
void GreenHouse::manageEmployee ( ) [private]
```

Manage the employee.

Definition at line 182 of file GreenHouse.cpp.

```
182
183     bool exit = false;
184     // Mostramos la ventana de employee
185     do {
186         sleep();
187         ms_->mainWindowEmployee();
188         switch (ms_->getSelection()) {
189             case 1:
190                 ms_->createAlarmScreen();
```

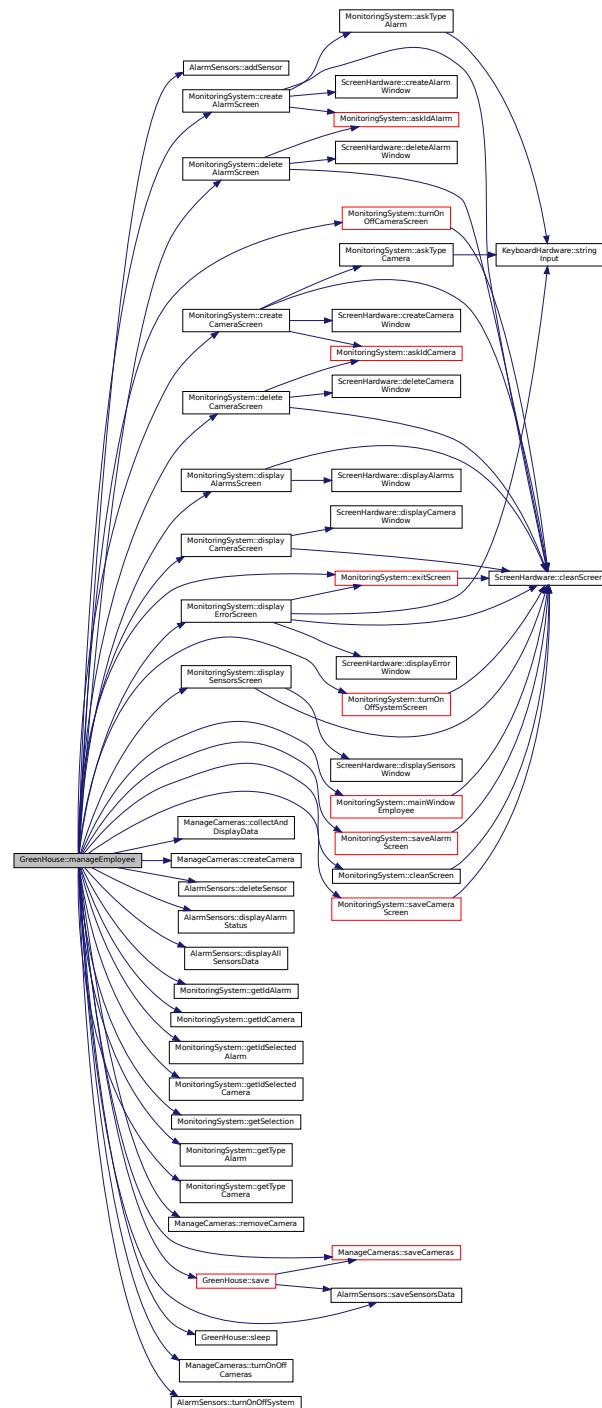
```

191     alarm_->addSensor(ms_->getIdAlarm(), ms_->getTypeAlarm());
192     break;
193 case 2:
194     ms_->deleteAlarmScreen();
195     alarm_->deleteSensor(ms_->getIdSelectedAlarm());
196     break;
197 case 3:
198     ms_->displaySensorsScreen();
199     alarm_->displayAllSensorsData();
200     break;
201 case 4:
202     ms_->displayAlarmsScreen();
203     alarm_->displayAlarmStatus();
204     break;
205 case 5:
206     ms_->turnOnOffSystemScreen();
207     alarm_->turnOnOffSystem(ms_->getSelection());
208     break;
209 case 6:
210     // Enseñar camaras
211     ms_->displayCameraScreen();
212     mc_->collectAndDisplayData();
213     break;
214 case 7:
215     // Crear camera
216     ms_->createCameraScreen();
217     mc_->createCamera(ms_->getIdCamera(), ms_->getTypeCamera());
218     break;
219 case 8:
220     // Borrar camera
221     ms_->deleteCameraScreen();
222     mc_->removeCamera(ms_->getIdSelectedCamera());
223     break;
224 case 9:
225     // Apagar o encender camaras
226     ms_->turnOnOffCameraScreen();
227     mc_->turnOnOffCameras(ms_->getSelection());
228     break;
229 case 10:
230     // Guardar camaras
231     ms_->saveCameraScreen();
232     mc_->saveCameras();
233     break;
234 case 11:
235     ms_->saveAlarmScreen();
236     alarm_->saveSensorsData();
237     break;
238 case 12:
239     ms_->cleanScreen();
240     save(2);
241     sleep();
242     ms_->exitScreen();
243     exit = true;
244     break;
245 default:
246     ms_->displayErrorScreen();
247     break;
248 }
249 } while (!exit);
250 }
```

References AlarmSensors::addSensor(), alarm\_, MonitoringSystem::cleanScreen(), ManageCameras::collectAndDisplayData(), MonitoringSystem::createAlarmScreen(), ManageCameras::createCamera(), MonitoringSystem::createCameraScreen(), MonitoringSystem::deleteAlarmScreen(), MonitoringSystem::deleteCameraScreen(), AlarmSensors::deleteSensor(), MonitoringSystem::displayAlarmsScreen(), AlarmSensors::displayAlarmStatus(), AlarmSensors::displayAllSensorsData(), MonitoringSystem::displayCameraScreen(), MonitoringSystem::displayErrorScreen(), MonitoringSystem::displaySensorsScreen(), MonitoringSystem::exitScreen(), MonitoringSystem::getIdAlarm(), MonitoringSystem::getIdCamera(), MonitoringSystem::getIdSelectedAlarm(), MonitoringSystem::getIdSelectedCamera(), MonitoringSystem::getSelection(), MonitoringSystem::getTypeAlarm(), MonitoringSystem::getTypeCamera(), MonitoringSystem::mainWindowEmployee(), mc\_, ms\_, ManageCameras::removeCamera(), save(), MonitoringSystem::saveAlarmScreen(), ManageCameras::saveCameras(), MonitoringSystem::saveCameraScreen(), AlarmSensors::saveSensorsData(), sleep(), ManageCameras::turnOnOffCameras(), MonitoringSystem::turnOnOffCameraScreen(), AlarmSensors::turnOnOffSystem(), and MonitoringSystem::turnOnOffSystemScreen().

Referenced by manageLogin().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.12.3.6 manageGuest()

```
void GreenHouse::manageGuest ( ) [private]
```

Manage the guest.

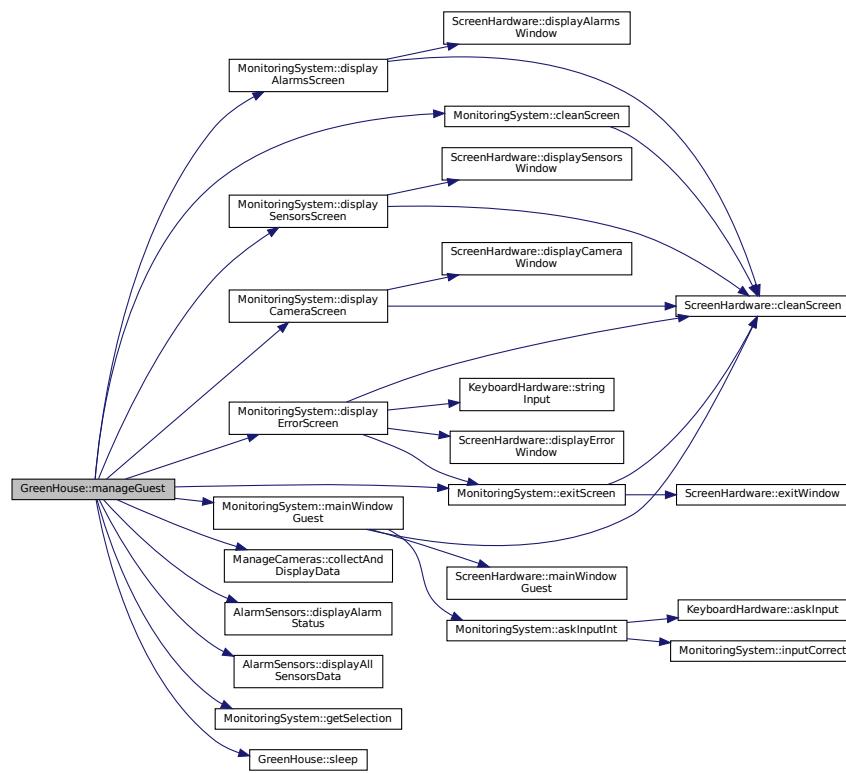
Definition at line 252 of file GreenHouse.cpp.

```
252     {
253     bool exit = false;
254     // Mostramos la ventana de guest
255     do {
256         sleep();
257         ms_>mainWindowGuest();
258         switch (ms_>getSelection()) {
259             case 1:
260                 ms_>displaySensorsScreen();
261                 alarm_>displayAllSensorsData();
262                 break;
263             case 2:
264                 ms_>displayAlarmsScreen();
265                 alarm_>displayAlarmStatus();
266                 break;
267             case 3:
268                 // Enseñar camaras
269                 ms_>displayCameraScreen();
270                 mc_>collectAndDisplayData();
271                 break;
272             case 4:
273                 ms_>cleanScreen();
274                 ms_>exitScreen();
275                 exit = true;
276                 break;
277             default:
278                 ms_>displayErrorScreen();
279                 break;
280         }
281     } while (!exit);
282 }
```

References alarm\_, MonitoringSystem::cleanScreen(), ManageCameras::collectAndDisplayData(), MonitoringSystem::displayAlarmsScreen(), AlarmSensors::displayAlarmStatus(), AlarmSensors::displayAllSensorsData(), MonitoringSystem::displayCameraScreen(), MonitoringSystem::displayErrorScreen(), MonitoringSystem::displaySensorsScreen(), MonitoringSystem::exitScreen(), MonitoringSystem::getSelection(), MonitoringSystem::mainWindowGuest(), mc\_, ms\_, and sleep().

Referenced by manageLogin().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.12.3.7 manageLogin()

```
void GreenHouse::manageLogin() [private]
```

Manage the login.

Definition at line 284 of file `GreenHouse.cpp`.

```

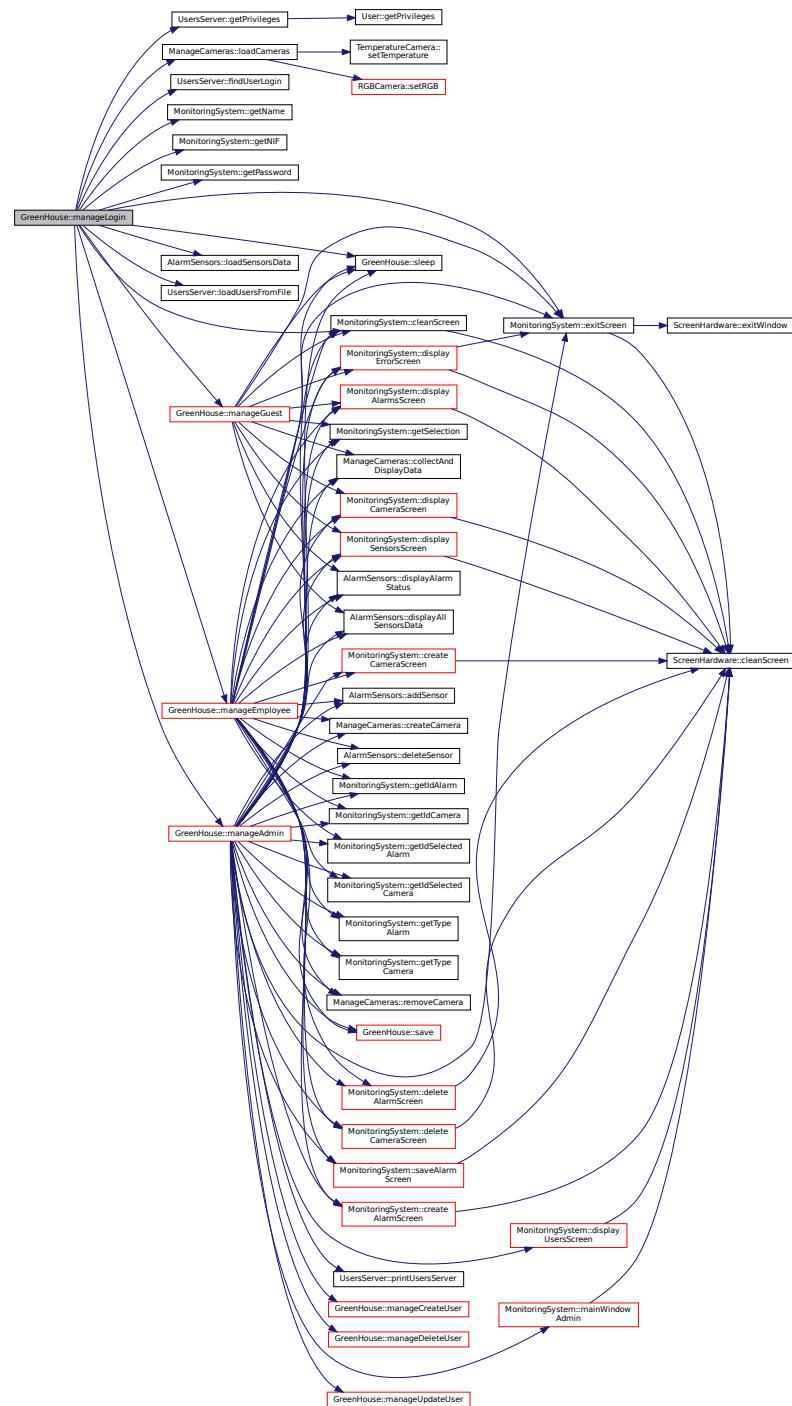
284
285
286 // Cargamos los usuarios del archivo
287 us_->loadUsersFromFile();
288 // Cargamos los sensores del archivo
289 alarm_->loadSensorsData();
290 // Cargamos las camaras del archivo
291 mc_->loadCameras();
292 sleep();
293 ms_->cleanScreen();
```

```
294 // Comprobamos si el usuario y la contraseña son correctos
295 if (us_->findUserLogin(ms_->getName(), ms_->getPassword(), ms_->getNIF())) {
296     printf("Usuario correcto\n");
297     // ahora tengo que ver que tipo de usuario es
298     // si es admin, employee o guest
299     // si es admin
300     if (us_->getPrivileges(ms_->getNIF()) == "ADMIN") {
301         manageAdmin();
302     } else if (us_->getPrivileges(ms_->getNIF()) == "EMPLOYEE") {
303         manageEmployee();
304     } else {
305         manageGuest();
306     }
307 } else {
308     printf("Usuario incorrecto\n");
309     sleep();
310     ms_->exitScreen();
311 }
312 }
```

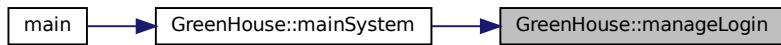
References alarm\_, MonitoringSystem::cleanScreen(), MonitoringSystem::exitScreen(), UsersServer::findUserLogin(), MonitoringSystem::getName(), MonitoringSystem::getNIF(), MonitoringSystem::getPassword(), UsersServer::getPrivileges(), ManageCameras::loadCameras(), AlarmSensors::loadSensorsData(), UsersServer::loadUsersFromFile(), manageAdmin(), manageEmployee(), manageGuest(), mc\_, ms\_, sleep(), and us\_.

Referenced by mainSystem().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.12.3.8 manageUpdateUser()

```
void GreenHouse::manageUpdateUser () [private]
```

Manage the update user.

Definition at line 65 of file GreenHouse.cpp.

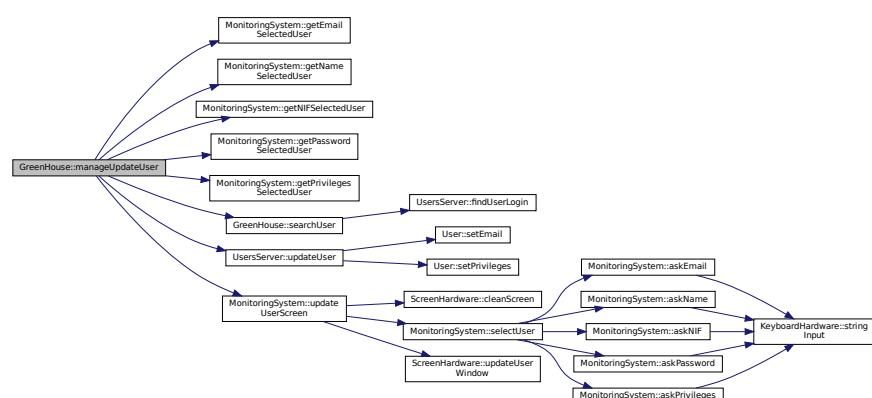
```

65
66     ms_->updateUserScreen ();
67     if (searchUser(ms_->getNameSelectedUser(), ms_->getPasswordSelectedUser(),
68                     ms_->getNIFSelectedUser()) ) {
69         us_->updateUser(ms_->getEmailSelectedUser(), ms_->getNIFSelectedUser(),
70                           ms_->getPasswordSelectedUser(),
71                           ms_->getPrivilegesSelectedUser(),
72                           ms_->getEmailSelectedUser());
73     } else {
74         printf("Usuario no encontrado\n");
75     }
76 }
```

References MonitoringSystem::getEmailSelectedUser(), MonitoringSystem::getNameSelectedUser(), MonitoringSystem::getNIFSelectedUser(), MonitoringSystem::getPasswordSelectedUser(), MonitoringSystem::getPrivilegesSelectedUser(), ms\_-, searchUser(), UsersServer::updateUser(), MonitoringSystem::updateUserScreen(), and us\_-.

Referenced by manageAdmin().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.12.3.9 save()

```
void GreenHouse::save (
    int id ) [private]
```

Save the system.

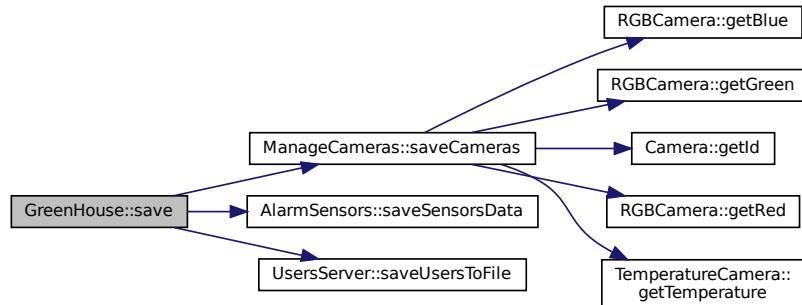
Definition at line 80 of file GreenHouse.cpp.

```
80
81     if (id == 1) {
82         us_->saveUsersToFile();
83         alarm_->saveSensorsData();
84         mc_->saveCameras();
85     } else if (id == 2) {
86         alarm_->saveSensorsData();
87         mc_->saveCameras();
88     }
89 }
```

References alarm\_, mc\_, ManageCameras::saveCameras(), AlarmSensors::saveSensorsData(), UsersServer::saveUsersToFile(), and us\_.

Referenced by manageAdmin(), and manageEmployee().

Here is the call graph for this function:



Here is the caller graph for this function:



### 4.12.3.10 searchUser()

```
bool GreenHouse::searchUser (
    std::string name,
    std::string password,
    std::string nif ) [private]
```

Manage search user.

Definition at line 42 of file GreenHouse.cpp.

```
43     {
44     return us_->findUserLogin(name, password, nif);
45 }
```

References UsersServer::findUserLogin(), and us\_-.

Referenced by manageDeleteUser(), and manageUpdateUser().

Here is the call graph for this function:



Here is the caller graph for this function:



### 4.12.3.11 sleep()

```
void GreenHouse::sleep () [private]
```

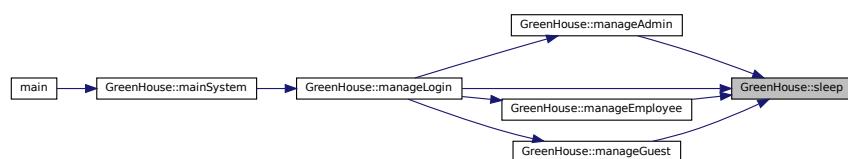
Sleep the system.

Definition at line 78 of file GreenHouse.cpp.

```
78 { system("sleep 2"); }
```

Referenced by manageAdmin(), manageEmployee(), manageGuest(), and manageLogin().

Here is the caller graph for this function:



#### 4.12.3.12 startSystem()

```
void GreenHouse::startSystem ( )
```

Start the system.

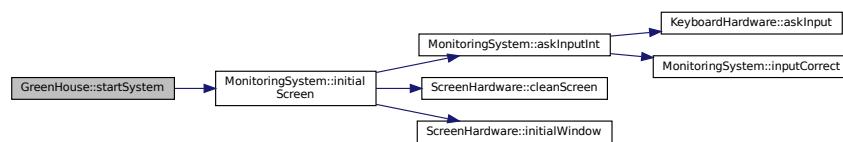
Definition at line 37 of file GreenHouse.cpp.

```
37      {
38      // Mensaje de bienvenida del invernadero
39      ms_->initialScreen();
40  }
```

References MonitoringSystem::initialScreen(), and ms\_-.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.12.4 Member Data Documentation

##### 4.12.4.1 alarm\_

```
AlarmSensors* GreenHouse::alarm_ [private]
```

This attribute is the `AlarmSensors` object (set of pointers to sensors).

Definition at line 99 of file GreenHouse.h.

Referenced by manageAdmin(), manageEmployee(), manageGuest(), manageLogin(), save(), and ~GreenHouse().

#### 4.12.4.2 mc\_

`ManageCameras* GreenHouse::mc_ [private]`

This attribute is the [ManageCameras](#) object (set of pointers to cameras).

Definition at line 118 of file `GreenHouse.h`.

Referenced by `manageAdmin()`, `manageEmployee()`, `manageGuest()`, `manageLogin()`, `save()`, and `~GreenHouse()`.

#### 4.12.4.3 ms\_

`MonitoringSystem* GreenHouse::ms_ [private]`

This attribute is the [MonitoringSystem](#) object.

Definition at line 105 of file `GreenHouse.h`.

Referenced by `mainSystem()`, `manageAdmin()`, `manageCreateUser()`, `manageDeleteUser()`, `manageEmployee()`, `manageGuest()`, `manageLogin()`, `manageUpdateUser()`, `startSystem()`, and `~GreenHouse()`.

#### 4.12.4.4 us\_

`UsersServer* GreenHouse::us_ [private]`

This attribute is the [UsersServer](#) object (set of pointers to users).

Definition at line 111 of file `GreenHouse.h`.

Referenced by `manageAdmin()`, `manageCreateUser()`, `manageDeleteUser()`, `manageLogin()`, `manageUpdateUser()`, `save()`, `searchUser()`, and `~GreenHouse()`.

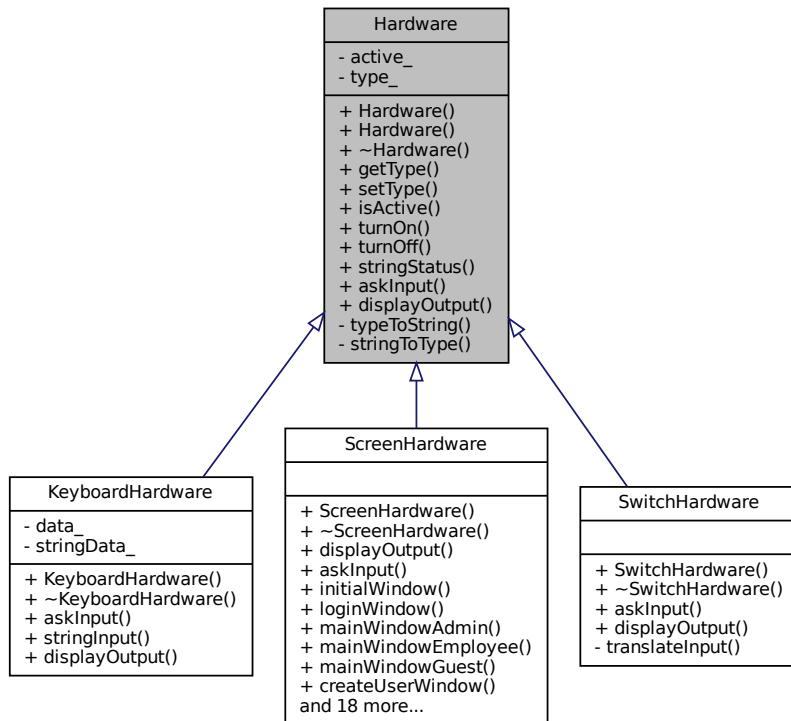
The documentation for this class was generated from the following files:

- src/[GreenHouse.h](#)
- src/[GreenHouse.cpp](#)

## 4.13 Hardware Class Reference

```
#include <Hardware.h>
```

Inheritance diagram for Hardware:



Collaboration diagram for Hardware:

Hardware
<code>- active_</code> <code>- type_</code>
<code>+ Hardware()</code> <code>+ Hardware()</code> <code>+ ~Hardware()</code> <code>+ getType()</code> <code>+ setType()</code> <code>+ isActive()</code> <code>+ turnOn()</code> <code>+ turnOff()</code> <code>+ stringStatus()</code> <code>+ askInput()</code> <code>+ displayOutput()</code> <code>- typeToString()</code> <code>- stringToType()</code>

## Public Types

- enum `Types_Hardware { NONE , SCREEN , KEYBOARD , SWITCH }`  
*Enum of the types of hardware.*

## Public Member Functions

- `Hardware ()`  
*Construct a new `Hardware` object.*
- `Hardware (bool active, Types_Hardware type)`  
*Construct a new `Hardware` object.*
- `virtual ~Hardware ()`  
*Destroy the `Hardware` object.*
- `std::string getType () const`  
*Get the Type object.*
- `void setType (std::string newtype)`  
*Set the Type object.*
- `bool isActive () const`  
*If the hardware is active.*
- `void turnOn ()`  
*Turn on the hardware.*
- `void turnOff ()`  
*Turn off the hardware.*
- `std::string stringStatus () const`  
*This method returns if the hardware is active or not in a string.*
- `virtual int askInput ()`  
*This method asks the user for an input.*
- `virtual void displayOutput () const`  
*This method displays the output of the hardware.*

## Private Member Functions

- std::string `typeToString (Types_Hardware type) const`  
*This method converts the type of hardware to a string.*
- `Types_Hardware stringToType (std::string type) const`  
*This method converts the string to a type of hardware.*

## Private Attributes

- bool `active_`  
*This attribute is the status of the hardware.*
- `Types_Hardware type_`  
*This attribute is the type of the hardware.*

### 4.13.1 Detailed Description

Definition at line 15 of file Hardware.h.

### 4.13.2 Member Enumeration Documentation

#### 4.13.2.1 Types\_Hardware

```
enum Hardware::Types_Hardware
```

Enum of the types of hardware.

Enumerator

NONE	
SCREEN	
KEYBOARD	
SWITCH	

Definition at line 21 of file Hardware.h.

```
21 { NONE, SCREEN, KEYBOARD, SWITCH };
```

### 4.13.3 Constructor & Destructor Documentation

#### 4.13.3.1 Hardware() [1/2]

```
Hardware::Hardware ( )
```

Construct a new [Hardware](#) object.

Definition at line 7 of file [Hardware.cpp](#).

```
7 : active_(false), type_(Types_Hardware::NONE) {}
```

#### 4.13.3.2 [Hardware\(\)](#) [2/2]

```
Hardware::Hardware (
    bool active,
    Types_Hardware type ) [explicit]
```

Construct a new [Hardware](#) object.

**Parameters**

<i>active</i>	
<i>type</i>	

Definition at line 8 of file [Hardware.cpp](#).

```
9 : active_(active), type_(type) {}
```

#### 4.13.3.3 [~Hardware\(\)](#)

```
Hardware::~Hardware ( ) [virtual]
```

Destroy the [Hardware](#) object.

Definition at line 10 of file [Hardware.cpp](#).

```
10 { }
```

### 4.13.4 Member Function Documentation

#### 4.13.4.1 [askInput\(\)](#)

```
int Hardware::askInput ( ) [virtual]
```

This method asks the user for an input.

**Returns**

```
int
```

Reimplemented in [SwitchHardware](#), [ScreenHardware](#), and [KeyboardHardware](#).

Definition at line 34 of file [Hardware.cpp](#).

```
34 {
35     return 0;
36     // esta funcion sera definida en clases hijas
37     // pero la idea es que muestre un mensaje estilo Pantalla: (Aqui viene el
38     // input del usuario)
39 }
```

#### 4.13.4.2 displayOutput()

```
void Hardware::displayOutput ( ) const [virtual]
```

This method displays the output of the hardware.

Reimplemented in [SwitchHardware](#), [ScreenHardware](#), and [KeyboardHardware](#).

Definition at line 41 of file [Hardware.cpp](#).

```
41  {
42  // Esta funcion sera definida en las clases hijas
43  // Por ahora mostramos un mensaje generico
44  cout << "Hardware cannot display output for this type" << endl;
45 }
```

#### 4.13.4.3 getType()

```
std::string Hardware::getType ( ) const
```

Get the Type object.

Returns

```
std::string
```

Definition at line 12 of file [Hardware.cpp](#).

```
12 { return typeToString(type_); }
```

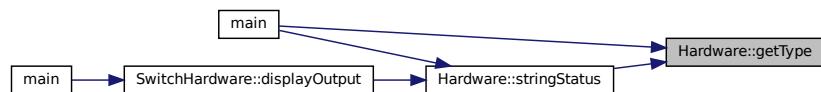
References `type_`, and `typeToString()`.

Referenced by `main()`, and `stringStatus()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.13.4.4 isActive()

```
bool Hardware::isActive () const
```

If the hardware is active.

##### Returns

true if the hardware is active  
false if the hardware is not active

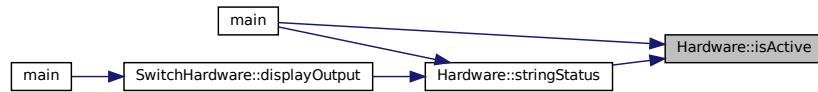
Definition at line 16 of file Hardware.cpp.

```
16 { return active_; }
```

References active\_.

Referenced by main(), and stringStatus().

Here is the caller graph for this function:



#### 4.13.4.5 setType()

```
void Hardware::setType ( std::string newtype )
```

Set the Type object.

##### Parameters

<code>newtype</code>	<input type="text"/>
----------------------	----------------------

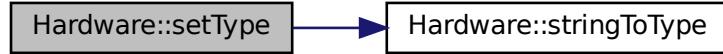
Definition at line 14 of file Hardware.cpp.

```
14 { type_ = stringToType(newtype); }
```

References `stringToType()`, and `type_`.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.13.4.6 stringStatus()

```
std::string Hardware::stringStatus ( ) const
```

This method returns if the hardware is active or not in a string.

##### Returns

```
std::string
```

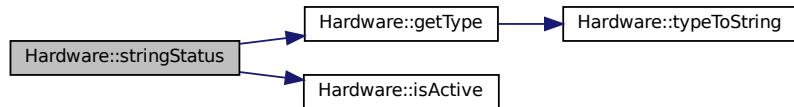
Definition at line 22 of file Hardware.cpp.

```
22
23     std::string status;
24     if (isActive()) {
25         status = "ON";
26     } else {
27         status = "OFF";
28     }
29     status += " - ";
30     status += getType();
31     return status;
32 }
```

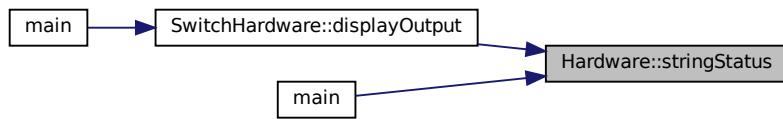
References `getType()`, and `isActive()`.

Referenced by `SwitchHardware::displayOutput()`, and `main()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.13.4.7 `stringToType()`

```
Hardware::Types_Hardware Hardware::stringToType (
    std::string type ) const [private]
```

This method converts the string to a type of hardware.

##### Parameters

<code>type</code>	<input type="text"/>
-------------------	----------------------

##### Returns

`Types_Hardware`

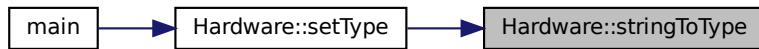
Definition at line 60 of file `Hardware.cpp`.

```

60
61     if (type == "SCREEN") {
62         return Hardware::Types_Hardware::SCREEN;
63     } else if (type == "KEYBOARD") {
64         return Hardware::Types_Hardware::KEYBOARD;
65     } else if (type == "SWITCH") {
66         return Hardware::Types_Hardware::SWITCH;
67     } else {
68         return Hardware::Types_Hardware::NONE;
69     }
70 }
```

Referenced by `setType()`.

Here is the caller graph for this function:



#### 4.13.4.8 turnOff()

```
void Hardware::turnOff ( )
```

Turn off the hardware.

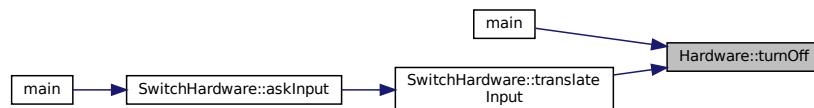
Definition at line 20 of file Hardware.cpp.

```
20 { active_ = false; }
```

References active\_.

Referenced by main(), and SwitchHardware::translateInput().

Here is the caller graph for this function:



#### 4.13.4.9 turnOn()

```
void Hardware::turnOn ( )
```

Turn on the hardware.

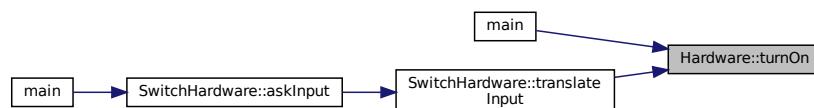
Definition at line 18 of file Hardware.cpp.

```
18 { active_ = true; }
```

References active\_.

Referenced by main(), and SwitchHardware::translateInput().

Here is the caller graph for this function:



#### 4.13.4.10 typeToString()

```
std::string Hardware::typeToString (
    Types_Hardware type ) const [private]
```

This method converts the type of hardware to a string.

##### Parameters

<i>type</i>	<input type="text"/>
-------------	----------------------

##### Returns

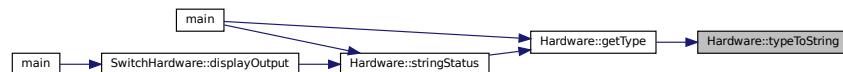
`std::string`

Definition at line 47 of file `Hardware.cpp`.

```
47
48     switch (type) {
49     case Types_Hardware::SCREEN:
50         return "SCREEN";
51     case Types_Hardware::KEYBOARD:
52         return "KEYBOARD";
53     case Types_Hardware::SWITCH:
54         return "SWITCH";
55     default:
56         return "None";
57     }
58 }
```

Referenced by `getType()`.

Here is the caller graph for this function:



#### 4.13.5 Member Data Documentation

##### 4.13.5.1 active\_

```
bool Hardware::active_ [private]
```

This attribute is the status of the hardware.

Definition at line 95 of file `Hardware.h`.

Referenced by `isActive()`, `turnOff()`, and `turnOn()`.

#### 4.13.5.2 type\_

`Types_Hardware Hardware::type_ [private]`

This attribute is the type of the hardware.

Definition at line 100 of file Hardware.h.

Referenced by `getType()`, and `setType()`.

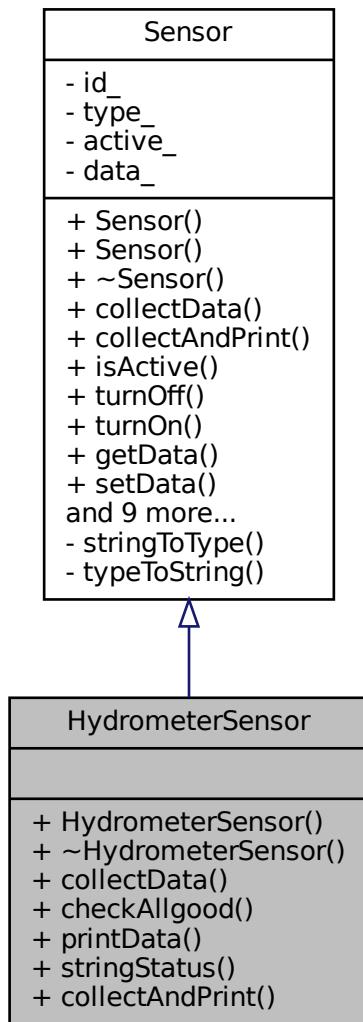
The documentation for this class was generated from the following files:

- src/[Hardware.h](#)
- src/[Hardware.cpp](#)

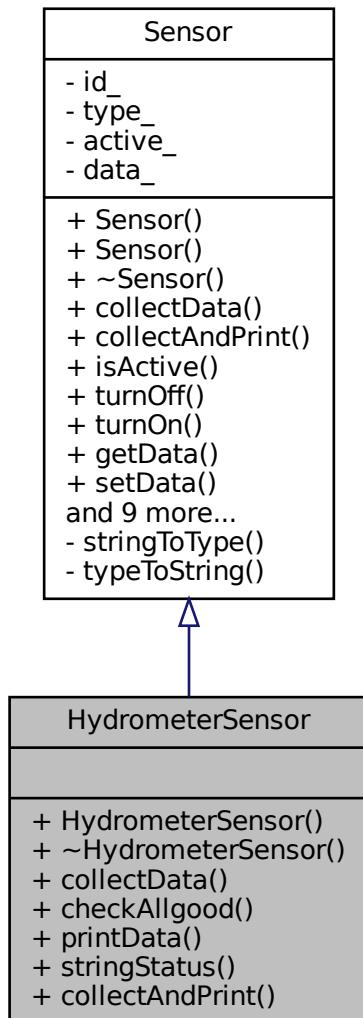
## 4.14 HydrometerSensor Class Reference

```
#include <HydrometerSensor.h>
```

Inheritance diagram for HydrometerSensor:



Collaboration diagram for HydrometerSensor:



## Public Member Functions

- `HydrometerSensor (int id, bool active)`  
*Construct a new Hydrometer `Sensor` object.*
- `~HydrometerSensor () override`  
*Destroy the Hydrometer `Sensor` object.*
- `void collectData () override`  
*Collect data of the Hydrometer `Sensor`.*
- `bool checkAllgood () const override`  
*Check if the Hydrometer `Sensor` is working properly.*
- `void printData () const override`  
*Print the data of the Hydrometer `Sensor`.*
- `std::string stringStatus () const`

*Collect and print the data of the Hydrometer Sensor.*

- void `collectAndPrint ()`

*Collect and print the data of the Hydrometer Sensor.*

## Friends

- std::ostream & `operator<< (std::ostream &os, const HydrometerSensor &sensor)`  
*Get the Data object.*

## Additional Inherited Members

### 4.14.1 Detailed Description

Definition at line 15 of file HydrometerSensor.h.

### 4.14.2 Constructor & Destructor Documentation

#### 4.14.2.1 HydrometerSensor()

```
HydrometerSensor::HydrometerSensor (
    int id,
    bool active ) [explicit]
```

Construct a new Hydrometer Sensor object.

#### Parameters

<code>id</code>	
<code>active</code>	

#### Returns

HydrometerSensor object

Definition at line 9 of file HydrometerSensor.cpp.

```
10     : Sensor(id, Sensor::Types::HYDROMETER, active) {}
```

#### 4.14.2.2 ~HydrometerSensor()

```
HydrometerSensor::~HydrometerSensor () [override]
```

Destroy the Hydrometer Sensor object.

Definition at line 12 of file HydrometerSensor.cpp.

```
12 { }
```

### 4.14.3 Member Function Documentation

#### 4.14.3.1 checkAllgood()

```
bool HydrometerSensor::checkAllgood ( ) const [override], [virtual]
```

Check if the Hydrometer Sensor is working properly.

##### Returns

true if the Hydrometer Sensor is working properly  
 false if the Hydrometer Sensor is not working properly

Reimplemented from [Sensor](#).

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

```
25   float data = Sensor::getData();
26   // Reading between 55-85 is considered good
27   if (data >= 52 && data <= 88) {
28     return true;
29   } else {
30     return false;
31   }
32 }
```

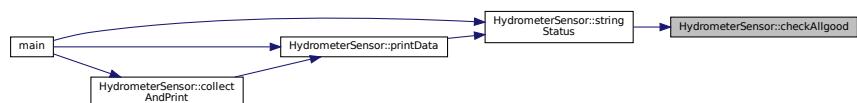
References [Sensor::getData\(\)](#).

Referenced by [stringStatus\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.14.3.2 collectAndPrint()

```
void HydrometerSensor::collectAndPrint ( ) [virtual]
```

Collect and print the data of the Hydrometer [Sensor](#).

Reimplemented from [Sensor](#).

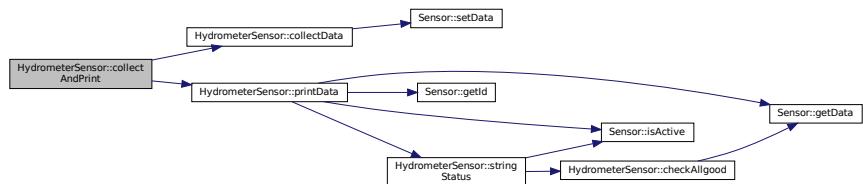
Definition at line 65 of file [HydrometerSensor.cpp](#).

```
65
66   collectData();
67   printData();
68 }
```

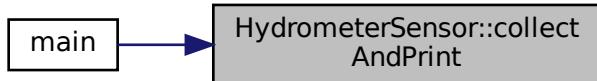
References [collectData\(\)](#), and [printData\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.14.3.3 collectData()

```
void HydrometerSensor::collectData ( ) [override], [virtual]
```

Collect data of the Hydrometer [Sensor](#).

This method collects the data of the Hydrometer [Sensor](#) and stores it in the data attribute.

Reimplemented from [Sensor](#).

Definition at line 14 of file HydrometerSensor.cpp.

```

14      {
15      // Generate random hydrometer reading between 50-90
16      std::random_device rd;
17      std::mt19937 gen(rd());
18      std::uniform_int_distribution<> dis(50, 90);
19      int reading = dis(gen);
20
21      // Set data
22      Sensor::setData(reading);
23  }
```

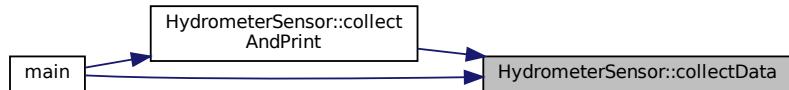
References Sensor::setData().

Referenced by collectAndPrint(), and main().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.14.3.4 printData()

```
void HydrometerSensor::printData ( ) const [override], [virtual]
```

Print the data of the Hydrometer Sensor.

Reimplemented from [Sensor](#).

Definition at line 52 of file HydrometerSensor.cpp.

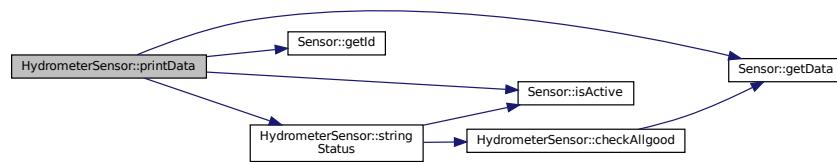
```

52      {
53      if (Sensor::isActive()) {
54          std::cout << "Hydrometer Sensor with "
55          << "ID: " << Sensor::getId() << " - Data: " << Sensor::getData()
56          << " % "
57          << "- Status: " << stringStatus() << endl;
58
59      } else {
60          cout << "Hydrometer Sensor ID: " << Sensor::getId() << " - INACTIVE"
61          << endl;
62      }
63  }
```

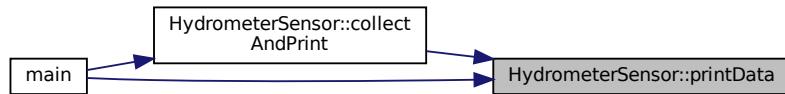
References `Sensor::getData()`, `Sensor::getId()`, `Sensor::isActive()`, and `stringStatus()`.

Referenced by `collectAndPrint()`, and `main()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.14.3.5 stringStatus()

```
std::string HydrometerSensor::stringStatus ( ) const
```

Collect and print the data of the Hydrometer [Sensor](#).

##### Returns

`std::string` of the status of the Hydrometer [Sensor](#)

Definition at line 40 of file `HydrometerSensor.cpp`.

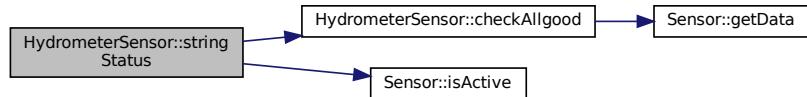
```

40
41     if (Sensor::isActive()) {
42         if (this->checkAllgood()) {
43             return "ACTIVE - GOOD STATUS";
44         } else {
45             return "ACTIVE - BAD STATUS";
46         }
47     } else {
48         return "INACTIVE";
49     }
50 }
```

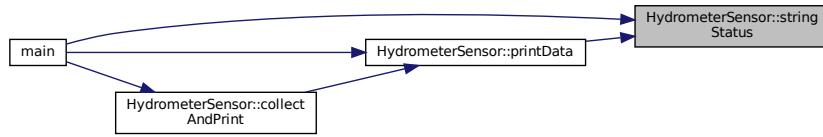
References `checkAllgood()`, and `Sensor::isActive()`.

Referenced by `main()`, and `printData()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.14.4 Friends And Related Function Documentation

##### 4.14.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const HydrometerSensor & sensor ) [friend]
```

Get the Data object.

**Returns**

double

Definition at line 35 of file HydrometerSensor.cpp.

```
35
36     sensor.printData();
37     return os;
38 }
```

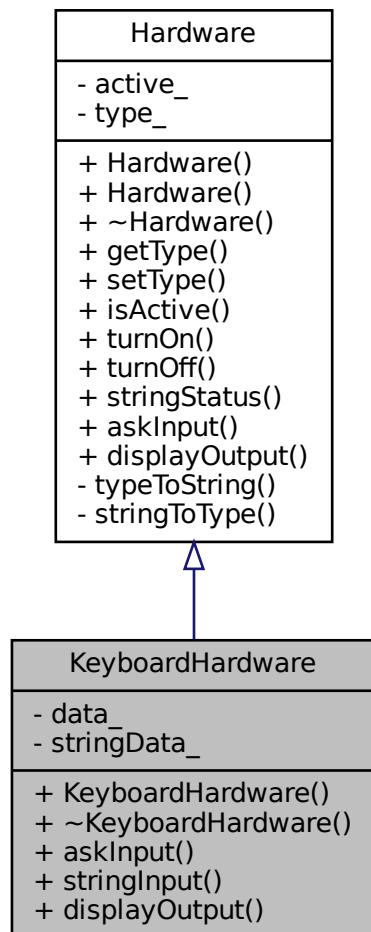
The documentation for this class was generated from the following files:

- [src/HydrometerSensor.h](#)
- [src/HydrometerSensor.cpp](#)

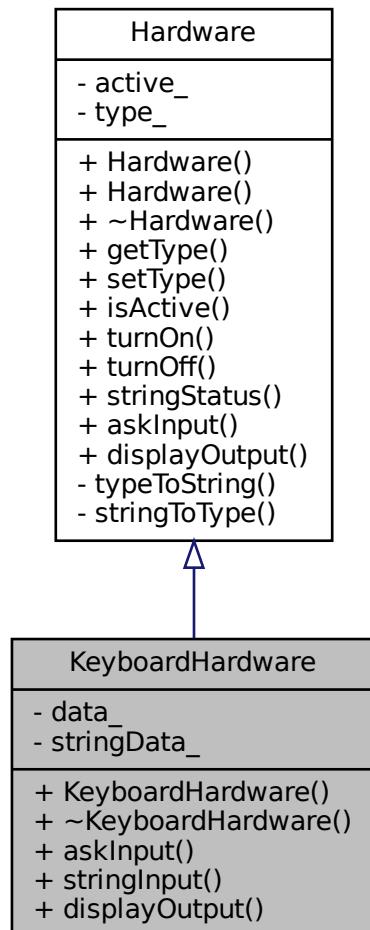
## 4.15 KeyboardHardware Class Reference

```
#include <KeyboardHardware.h>
```

Inheritance diagram for KeyboardHardware:



Collaboration diagram for KeyboardHardware:



## Public Member Functions

- `KeyboardHardware (bool active)`  
*Construct a new Keyboard `Hardware` object.*
- `~KeyboardHardware () override`  
*Destroy the Keyboard `Hardware` object.*
- int `askInput () override`  
*Ask for an input to the user.*
- std::string `stringInput ()`  
*Ask for a string input to the user.*
- void `displayOutput () const override`  
*Display the output of the Keyboard `Hardware`.*

## Private Attributes

- int `data_`  
*The int data of the Keyboard [Hardware](#).*
- std::string `stringData_`  
*The string data of the Keyboard [Hardware](#).*

## Additional Inherited Members

### 4.15.1 Detailed Description

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

### 4.15.2 Constructor & Destructor Documentation

#### 4.15.2.1 `KeyboardHardware()`

```
KeyboardHardware::KeyboardHardware (
    bool active ) [explicit]
```

Construct a new Keyboard [Hardware](#) object.

##### Parameters

<code>active</code>	<input type="checkbox"/>
---------------------	--------------------------

##### Returns

[KeyboardHardware](#) object

Definition at line 9 of file `KeyboardHardware.cpp`.

```
10     : Hardware(active, Hardware::Types_Hardware::KEYBOARD) {
11     data_ = 0;
12     stringData_ = "";
13 }
```

References `data_`, and `stringData_`.

#### 4.15.2.2 `~KeyboardHardware()`

```
KeyboardHardware::~KeyboardHardware () [override]
```

Destroy the Keyboard [Hardware](#) object.

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

```
15 { }
```

### 4.15.3 Member Function Documentation

#### 4.15.3.1 askInput()

```
int KeyboardHardware::askInput ( ) [override], [virtual]
```

Ask for an input to the user.

Returns

int

Reimplemented from [Hardware](#).

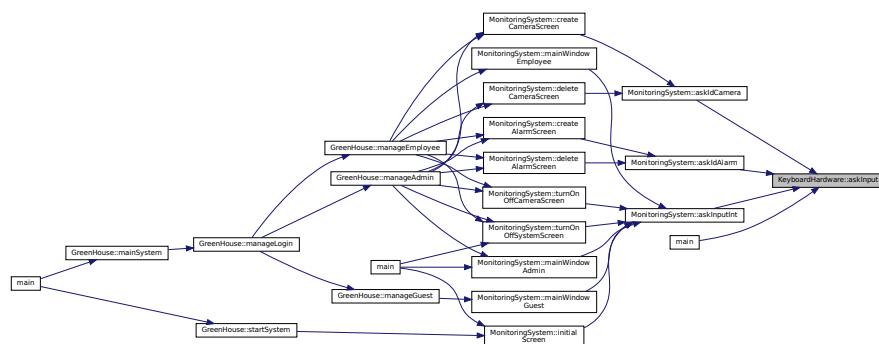
Definition at line 17 of file `KeyboardHardware.cpp`.

```
17
18     bool exit = false;
19     int input;
20
21     while (not exit) {
22         std::cout << "- Keyboard waiting for input (integer): ";
23         std::cin >> input;
24
25         // Verificar si la entrada es un número
26         if (std::cin.fail()) {
27             std::cin.clear(); // Restablecer el estado de cin
28             std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
29                             '\n'); // Limpiar el buffer de entrada
29             std::cout << "Invalid input. Please enter an integer that corresponds to "
30                     "one of the options"
31                     << std::endl;
32         } else {
33             exit = true;
34             data_ = input; // Guardar el valor ingresado en data_
35         }
36     }
37
38     return data_;
39 }
```

References `data_`.

Referenced by `MonitoringSystem::askIdAlarm()`, `MonitoringSystem::askIdCamera()`, `MonitoringSystem::askInput←Int()`, and `main()`.

Here is the caller graph for this function:



#### 4.15.3.2 displayOutput()

```
void KeyboardHardware::displayOutput ( ) const [override], [virtual]
```

Display the output of the Keyboard Hardware.

Reimplemented from [Hardware](#).

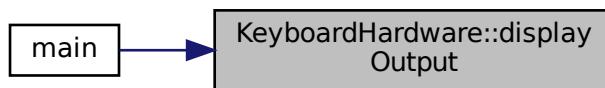
Definition at line 50 of file [KeyboardHardware.cpp](#).

```
50   std::cout << "-Last input(integer) of the keyboard: " << data_ << std::endl;
51   std::cout << "-Last input(string) of the keyboard: " << stringData_
52   << std::endl;
53 }
54 }
```

References `data_`, and `stringData_`.

Referenced by `main()`.

Here is the caller graph for this function:



#### 4.15.3.3 stringInput()

```
std::string KeyboardHardware::stringInput ( )
```

Ask for a string input to the user.

Returns

`std::string`

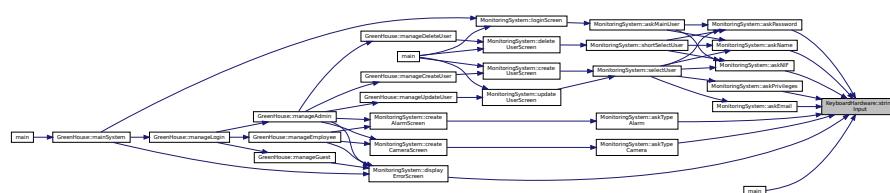
Definition at line 42 of file [KeyboardHardware.cpp](#).

```
42   std::cout << "- Keyboard waiting for input (string): ";
43   std::string input;
44   std::cin >> input;
45   stringData_ = input;
46   return stringData_;
47 }
48 }
```

References `stringData_`.

Referenced by `MonitoringSystem::askEmail()`, `MonitoringSystem::askName()`, `MonitoringSystem::askNIF()`, `MonitoringSystem::askPassword()`, `MonitoringSystem::askPrivileges()`, `MonitoringSystem::askTypeAlarm()`, `MonitoringSystem::askTypeCamera()`, `MonitoringSystem::displayErrorScreen()`, and `main()`.

Here is the caller graph for this function:



#### 4.15.4 Member Data Documentation

##### 4.15.4.1 **data\_**

```
int KeyboardHardware::data_ [private]
```

The int data of the Keyboard [Hardware](#).

Definition at line 52 of file KeyboardHardware.h.

Referenced by askInput(), displayOutput(), and KeyboardHardware().

##### 4.15.4.2 **stringData\_**

```
std::string KeyboardHardware::stringData_ [private]
```

The string data of the Keyboard [Hardware](#).

Definition at line 57 of file KeyboardHardware.h.

Referenced by displayOutput(), KeyboardHardware(), and stringInput().

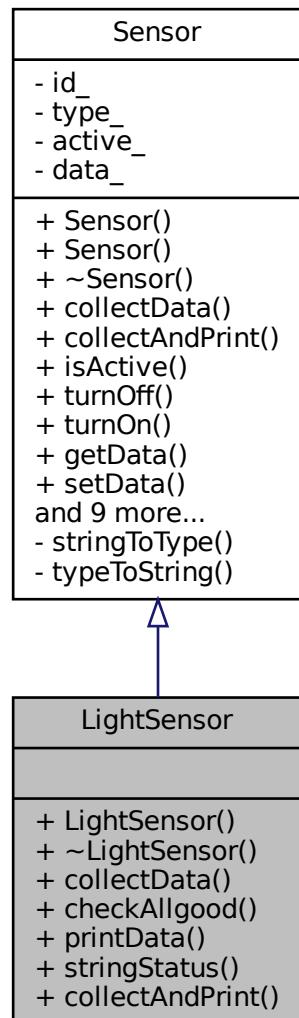
The documentation for this class was generated from the following files:

- src/[KeyboardHardware.h](#)
- src/[KeyboardHardware.cpp](#)

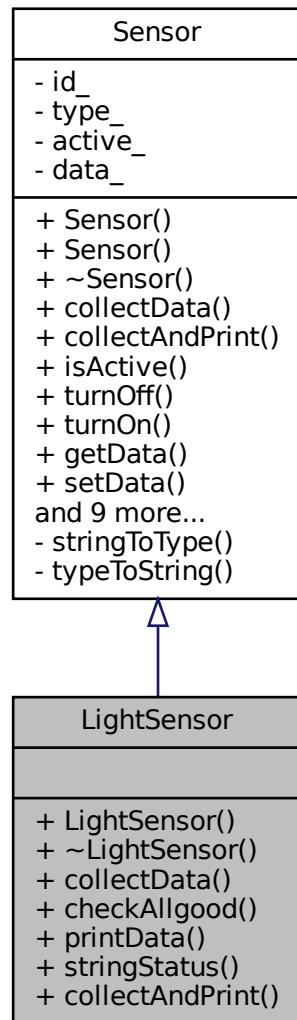
## 4.16 LightSensor Class Reference

```
#include <LightSensor.h>
```

Inheritance diagram for LightSensor:



Collaboration diagram for LightSensor:



## Public Member Functions

- `LightSensor (int id, bool active)`  
*Construct a new Light Sensor object.*
- `~LightSensor () override`  
*Destroy the Light Sensor object.*
- `void collectData () override`  
*Collect data of the Light Sensor.*
- `bool checkAllgood () const override`  
*Check if the Light Sensor is working properly.*
- `void printData () const override`  
*Print the data of the Light Sensor.*
- `std::string stringStatus () const`

*Collect and print the data of the Light Sensor.*

- void `collectAndPrint ()`

*Collect and print the data of the Light Sensor.*

## Friends

- std::ostream & `operator<< (std::ostream &os, const LightSensor &sensor)`

*Get the Light object.*

## Additional Inherited Members

### 4.16.1 Detailed Description

Definition at line 15 of file LightSensor.h.

### 4.16.2 Constructor & Destructor Documentation

#### 4.16.2.1 LightSensor()

```
LightSensor::LightSensor (
    int id,
    bool active ) [explicit]
```

Construct a new Light Sensor object.

##### Parameters

<code>id</code>	
<code>active</code>	

##### Returns

`LightSensor` object

Definition at line 8 of file LightSensor.cpp.

```
9     : Sensor(id, Sensor::Types::LIGHT_SENSOR, active) {}
```

#### 4.16.2.2 ~LightSensor()

```
LightSensor::~LightSensor () [override]
```

Destroy the Light Sensor object.

Definition at line 11 of file LightSensor.cpp.

```
11 { }
```

### 4.16.3 Member Function Documentation

#### 4.16.3.1 checkAllgood()

```
bool LightSensor::checkAllgood ( ) const [override], [virtual]
```

Check if the Light Sensor is working properly.

##### Returns

true if the Light Sensor is working properly  
false if the Light Sensor is not working properly

Reimplemented from [Sensor](#).

Definition at line 23 of file [LightSensor.cpp](#).

```
23
24     float data = Sensor::getData();
25
26     if (data >= 300 && data <= 3900) {
27         return true;
28     } else {
29         return false;
30     }
31 }
```

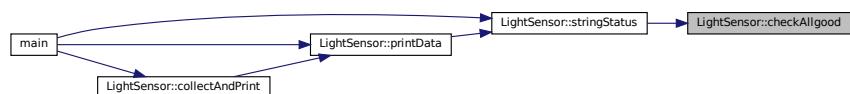
References [Sensor::getData\(\)](#).

Referenced by [stringStatus\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.16.3.2 collectAndPrint()

```
void LightSensor::collectAndPrint ( ) [virtual]
```

Collect and print the data of the Light [Sensor](#).

Reimplemented from [Sensor](#).

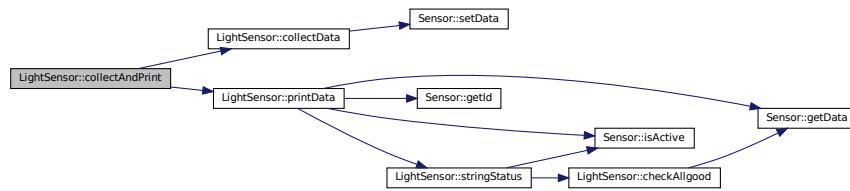
Definition at line 61 of file LightSensor.cpp.

```
61
62   collectData();
63   printData();
64 }
```

References [collectData\(\)](#), and [printData\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.16.3.3 collectData()

```
void LightSensor::collectData ( ) [override], [virtual]
```

Collect data of the Light [Sensor](#).

This method collects the data of the Light [Sensor](#) and stores it in the data attribute.

Reimplemented from [Sensor](#).

Definition at line 13 of file LightSensor.cpp.

```
13 {
```

```

14 // Medido en lux entre 200-4000 lux
15 std::random_device rd;
16 std::mt19937 gen(rd());
17 std::uniform_int_distribution<> dis(200, 4000);
18 int reading = dis(gen);
19
20 Sensor::setData(reading);
21 }

```

References [Sensor::setData\(\)](#).

Referenced by [collectAndPrint\(\)](#), and [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.16.3.4 printData()

```
void LightSensor::printData ( ) const [override], [virtual]
```

Print the data of the Light [Sensor](#).

Reimplemented from [Sensor](#).

Definition at line 38 of file [LightSensor.cpp](#).

```

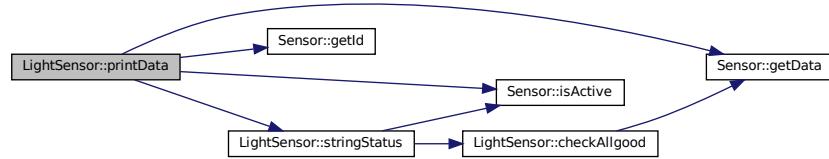
38 {
39     if (Sensor::isActive\(\)) {
40         std::cout << "Light Sensor with "
41             << "ID: " << Sensor::getId\(\) << " - Data: " << Sensor::getData\(\)
42             << " lux - Status: " << stringStatus\(\) << endl;
43     } else {
44         std::cout << "Light Sensor ID: " << Sensor::getId\(\)
45             << " - Status: " << stringStatus\(\) << endl;
46     }
47 }

```

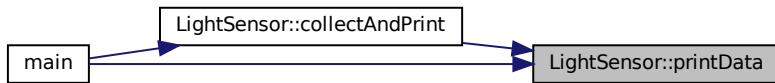
References [Sensor::getData\(\)](#), [Sensor::getId\(\)](#), [Sensor::isActive\(\)](#), and [stringStatus\(\)](#).

Referenced by `collectAndPrint()`, and `main()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.16.3.5 `stringStatus()`

```
std::string LightSensor::stringStatus () const
```

Collect and print the data of the Light [Sensor](#).

Definition at line 49 of file `LightSensor.cpp`.

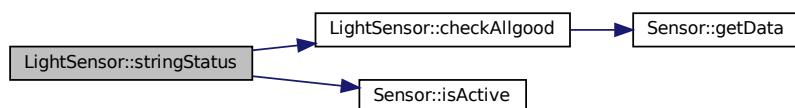
```

49
50     if (Sensor::isActive ()) {
51         if (this->checkAllgood ()) {
52             return "ACTIVE - GOOD STATUS";
53         } else {
54             return "ACTIVE - BAD STATUS";
55         }
56     } else {
57         return "INACTIVE";
58     }
59 }
```

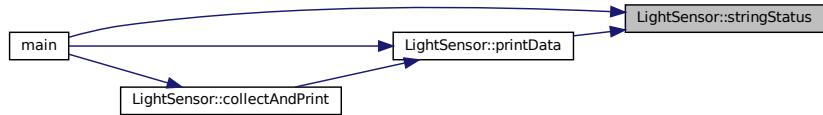
References `checkAllgood()`, and `Sensor::isActive()`.

Referenced by `main()`, and `printData()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.16.4 Friends And Related Function Documentation

##### 4.16.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const LightSensor & sensor ) [friend]
```

Get the Light object.

###### Returns

int

Definition at line 33 of file LightSensor.cpp.

```
33
34     sensor.printData();
35     return os;
36 }
```

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

- src/[LightSensor.h](#)
- src/[LightSensor.cpp](#)

## 4.17 ManageCameras Class Reference

```
#include <ManageCameras.h>
```

Collaboration diagram for ManageCameras:

ManageCameras
- cameras_ - filename_
+ ManageCameras() + ~ManageCameras() + createCamera() + clearCameras() + addCamera() + removeCamera() + findCamera() + displayAllCameras() + collectAndDisplayData() + collectData() + turnOnCameras() + turnOffCameras() + turnOnOffCameras() + saveCameras() + loadCameras()

## Public Member Functions

- **ManageCameras ()**  
*Construct a new Manage Cameras object.*
- **~ManageCameras ()**  
*Destroy the Manage Cameras object.*
- **void createCamera (int id, std::string type)**  
*Create a Camera object.*
- **void clearCameras ()**  
*Clear all the cameras.*
- **void addCamera (Camera \*camera)**  
*Add a Camera object.*
- **void removeCamera (int id)**  
*Remove a Camera object.*
- **Camera \* findCamera (int id)**  
*Find a Camera object.*
- **void displayAllCameras ()**  
*Display all the cameras.*
- **void collectAndDisplayData ()**  
*Collect and display the data of all the cameras.*
- **void collectData ()**  
*Collect the data of all the cameras.*
- **void turnOnCameras ()**  
*Turn on all the cameras.*

- void [turnOffCameras \(\)](#)  
*Turn off all the cameras.*
- void [turnOnOffCameras \(int status\)](#)  
*Turn on or off the cameras.*
- void [saveCameras \(\)](#)  
*Save the cameras to a file.*
- void [loadCameras \(\)](#)  
*Load the cameras from a file.*

## Private Attributes

- std::set< [Camera \\*](#) > [cameras\\_](#)  
*This attribute is a set of [Camera](#) pointers.*
- std::string [filename\\_](#) = "cameras.txt"  
*This attribute is the name of the file where the cameras are saved.*

### 4.17.1 Detailed Description

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

### 4.17.2 Constructor & Destructor Documentation

#### 4.17.2.1 [ManageCameras\(\)](#)

[ManageCameras](#)::[ManageCameras](#) ( )

Construct a new [Manage Cameras](#) object.

Definition at line 11 of file [ManageCameras.cpp](#).

```
11   {
12     // Crear las camaras por defecto
13     cameras_.insert(new RGBCamera(1));
14     cameras_.insert(new TemperatureCamera(2));
15 }
```

#### 4.17.2.2 [~ManageCameras\(\)](#)

[ManageCameras](#)::[~ManageCameras](#) ( )

Destroy the [Manage Cameras](#) object.

Definition at line 17 of file [ManageCameras.cpp](#).

```
17 { clearCameras(); }
```

### 4.17.3 Member Function Documentation

#### 4.17.3.1 [addCamera\(\)](#)

```
void ManageCameras::addCamera (
    Camera * camera )
```

Add a [Camera](#) object.

**Parameters**

<code>camera</code>	<input type="text"/>
---------------------	----------------------

Definition at line 46 of file ManageCameras.cpp.

```
46 { cameras_.insert(camera); }
```

#### 4.17.3.2 `clearCameras()`

```
void ManageCameras::clearCameras ()
```

Clear all the cameras.

Definition at line 19 of file ManageCameras.cpp.

```
19
20   for (auto camera : cameras_) {
21     delete camera;
22   }
23   cameras_.clear();
24 }
```

#### 4.17.3.3 `collectAndDisplayData()`

```
void ManageCameras::collectAndDisplayData ()
```

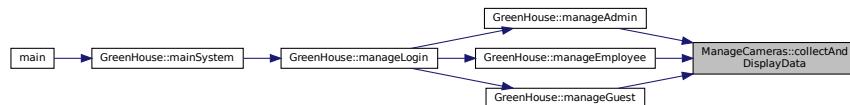
Collect and display the data of all the cameras.

Definition at line 83 of file ManageCameras.cpp.

```
83
84   collectData();
85   displayAllCameras();
86 }
```

Referenced by `GreenHouse::manageAdmin()`, `GreenHouse::manageEmployee()`, and `GreenHouse::manageGuest()`.

Here is the caller graph for this function:



#### 4.17.3.4 collectData()

```
void ManageCameras::collectData ( )
```

Collect the data of all the cameras.

Definition at line 77 of file ManageCameras.cpp.

```
77     {
78     for (Camera *camera : cameras_) {
79         camera->collectData();
80     }
81 }
```

Referenced by main().

Here is the caller graph for this function:



#### 4.17.3.5 createCamera()

```
void ManageCameras::createCamera (
    int id,
    std::string type )
```

Create a [Camera](#) object.

##### Parameters

<i>id</i>	
<i>type</i>	

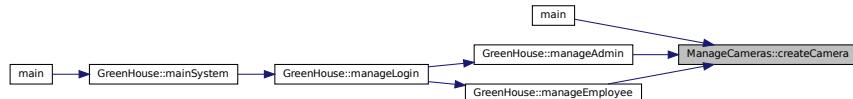
Definition at line 26 of file ManageCameras.cpp.

```
26
27 // Pasar a mayusculas el tipo
28 for (char &c : type) {
29     c = toupper(c);
30 }
31 // Crear la camara y añadir el puntero a cameras_
32 // SI ya esta ese id, no se crea la camara y se informa
33 if (findCamera(id) == nullptr) {
34     if (type == "RGB") {
35         cameras_.insert(new RGBCamera(id));
36     } else if (type == "TEMPERATURE") {
37         cameras_.insert(new TemperatureCamera(id));
38     } else {
39         cout << "Invalid camera type" << endl;
40     }
41 } else {
42     cout << "Camera with id " << id << " already exists" << endl;
```

```
43     }
44 }
```

Referenced by main(), GreenHouse::manageAdmin(), and GreenHouse::manageEmployee().

Here is the caller graph for this function:



#### 4.17.3.6 displayAllCameras()

```
void ManageCameras::displayAllCameras( )
```

Display all the cameras.

Definition at line 71 of file ManageCameras.cpp.

```
71 {
72     for (Camera *camera : cameras_) {
73         camera->printCamera();
74     }
75 }
```

Referenced by main().

Here is the caller graph for this function:



#### 4.17.3.7 findCamera()

```
Camera * ManageCameras::findCamera(
        int id)
```

Find a Camera object.

**Parameters**

<i>id</i>	
-----------	--

**Returns**

Camera\*

Definition at line 61 of file ManageCameras.cpp.

```

61      {
62      // Encontrar la camara con el id dado y retornar el puntero
63      for (Camera *camera : cameras_) {
64          if (camera->getId() == id) {
65              return camera;
66          }
67      }
68      return nullptr;
69 }
```

**4.17.3.8 loadCameras()**

void ManageCameras::loadCameras ( )

Load the cameras from a file.

Definition at line 142 of file ManageCameras.cpp.

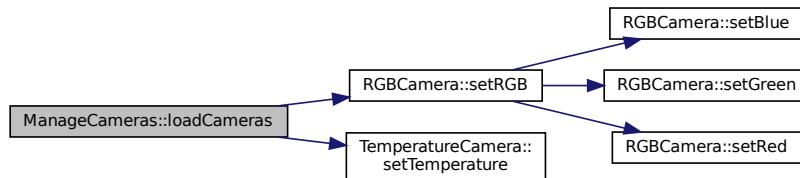
```

142      {
143      // Crear un archivo de entrada
144      std::ifstream file(filename_);
145      if (file.is_open()) {
146          // Hay que recorrer cada linea del archivo
147          // Primero se leera el id, luego el tipo y luego los datos rgb o temperatura
148          // dependiendo del tipo
149          // Primero borrar las camaras actuales
150          clearCameras();
151          while (!file.eof()) {
152              int id;
153              std::string type;
154              file >> id >> type;
155
156              if (type == "RGB") {
157                  int red, green, blue;
158                  file >> red >> green >> blue;
159                  if (findCamera(id) != nullptr) {
160                      cout << "Camera with id " << id << " already exists" << endl;
161                      continue; // Si ya existe la camara, no se crea y se pasa a la
162                      // siguiente
163                  }
164                  RGBCamera *rgbCamera = new RGBCamera(id);
165                  rgbCamera->setRGB(red, green, blue);
166                  cameras_.insert(rgbCamera);
167              } else if (type == "TEMPERATURE") {
168                  float temperature;
169                  file >> temperature;
170                  if (findCamera(id) != nullptr) {
171                      cout << "Camera with id " << id << " already exists" << endl;
172                      continue; // Si ya existe la camara, no se crea y se pasa a la
173                      // siguiente
174                  }
175                  TemperatureCamera *temperatureCamera = new TemperatureCamera(id);
176                  temperatureCamera->setTemperature(temperature);
177                  cameras_.insert(temperatureCamera);
178              }
179          }
180          file.close();
181          std::cout << "Cameras loaded" << std::endl;
182      } else {
183          std::cout << "Unable to open file" << std::endl;
184      }
185 }
```

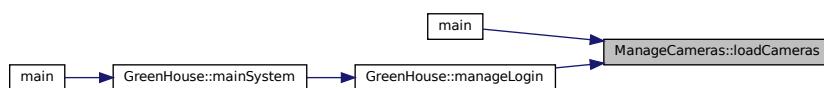
References RGBCamera::setRGB(), and TemperatureCamera::setTemperature().

Referenced by main(), and GreenHouse::manageLogin().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.17.3.9 removeCamera()

```
void ManageCameras::removeCamera (
    int id )
```

Remove a [Camera](#) object.

##### Parameters

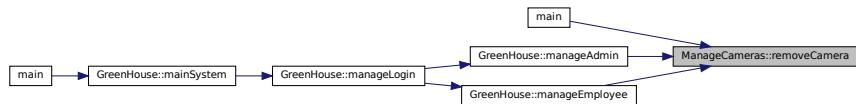
<code>id</code>	<input type="text"/>
-----------------	----------------------

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

```
48
49 // Usar findcamera para encontrar la camara con el id dado
50 Camera *camera = findCamera(id);
51 // Si la camara existe, borrarla
52 if (camera != nullptr) {
53     cout << "Removing camera with id " << id << endl;
54     cameras_.erase(camera);
55     delete camera;
56 } else {
57     cout << "Camera with id " << id << " does not exist" << endl;
58 }
59 }
```

Referenced by main(), GreenHouse::manageAdmin(), and GreenHouse::manageEmployee().

Here is the caller graph for this function:



#### 4.17.3.10 saveCameras()

```
void ManageCameras::saveCameras ( )
```

Save the cameras to a file.

Definition at line 112 of file ManageCameras.cpp.

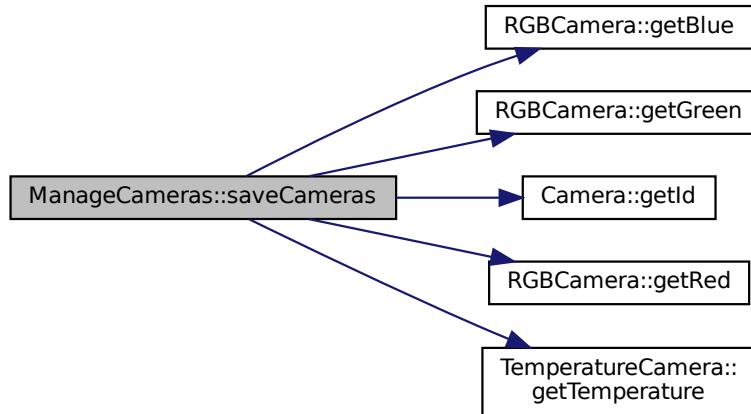
```

112
113 // Crear un archivo de salida
114 std::ofstream file(filename_);
115 if (file.is_open()) {
116     for (Camera *camera : cameras_) {
117         // Guardar el id, tipo y sus datos rgb o temperatura dependiendo del tipo
118         // Guardando en este orden las cosa TIPO ID Datos
119         if (camera->getType() == "RGB") {
120             std::cout << "Saving RGB camera (" << camera->getId() << ")"
121             << std::endl;
122             RGBCamera *rgbCamera = static_cast<RGBCamera *>(camera);
123             file << camera->getId() << " " << camera->getType() << " "
124             << rgbCamera->getRed() << " " << rgbCamera->getGreen() << " "
125             << rgbCamera->getBlue() << std::endl;
126         } else if (camera->getType() == "TEMPERATURE") {
127             std::cout << "Saving Temperature camera (" << camera->getId() << ")"
128             << std::endl;
129             TemperatureCamera *temperatureCamera =
130                 static_cast<TemperatureCamera *>(camera);
131             file << camera->getId() << " " << camera->getType() << " "
132             << temperatureCamera->getTemperature() << std::endl;
133         }
134     }
135     file.close();
136     std::cout << "Cameras saved" << std::endl;
137 } else {
138     std::cout << "Unable to open file" << std::endl;
139 }
140 }
```

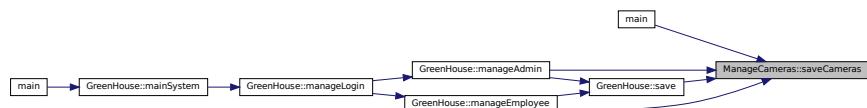
References RGBCamera::getBlue(), RGBCamera::getGreen(), Camera::getId(), RGBCamera::getRed(), and TemperatureCamera::getTemperature().

Referenced by main(), GreenHouse::manageAdmin(), GreenHouse::manageEmployee(), and GreenHouse::save().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.17.3.11 `turnOffCameras()`

```
void ManageCameras::turnOffCameras ( )
```

Turn off all the cameras.

Definition at line 95 of file `ManageCameras.cpp`.

```
95   {
96     cout << "Turning off the cameras...\n" << endl;
97     for (Camera *camera : cameras_) {
98       camera->turnOff();
99     }
100 }
```

Referenced by `main()`.

Here is the caller graph for this function:



#### 4.17.3.12 turnOnCameras()

```
void ManageCameras::turnOnCameras ( )
```

Turn on all the cameras.

Definition at line 88 of file ManageCameras.cpp.

```
88
89   cout << "Turning on the cameras...\n" << endl;
90   for (Camera *camera : cameras_) {
91     camera->turnOn();
92   }
93 }
```

#### 4.17.3.13 turnOnOffCameras()

```
void ManageCameras::turnOnOffCameras (
    int status )
```

Turn on or off the cameras.

**Parameters**

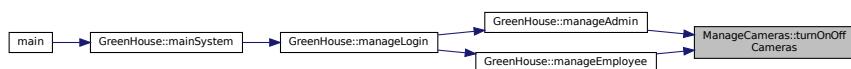
status	<input type="text"/>
--------	----------------------

Definition at line 102 of file ManageCameras.cpp.

```
102
103   if (status == 1) {
104     turnOnCameras();
105   } else if (status == 2) {
106     turnOffCameras();
107   } else {
108     cout << "Invalid status" << endl;
109   }
110 }
```

Referenced by GreenHouse::manageAdmin(), and GreenHouse::manageEmployee().

Here is the caller graph for this function:



#### 4.17.4 Member Data Documentation

#### 4.17.4.1 cameras\_

```
std::set<Camera *> ManageCameras::cameras_ [private]
```

This attribute is a set of [Camera](#) pointers.

Definition at line 116 of file [ManageCameras.h](#).

#### 4.17.4.2 filename\_

```
std::string ManageCameras::filename_ = "cameras.txt" [private]
```

This attribute is the name of the file where the cameras are saved.

Definition at line 121 of file [ManageCameras.h](#).

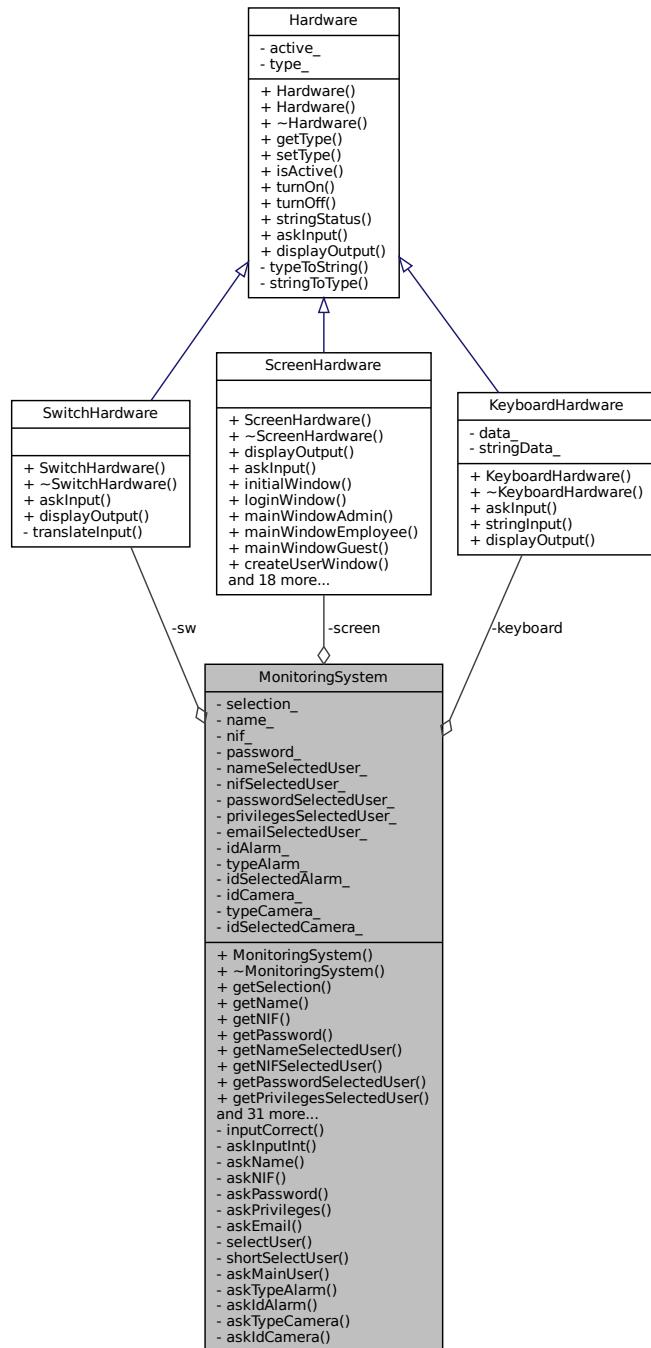
The documentation for this class was generated from the following files:

- src/[ManageCameras.h](#)
- src/[ManageCameras.cpp](#)

## 4.18 MonitoringSystem Class Reference

```
#include <MonitoringSystem.h>
```

Collaboration diagram for MonitoringSystem:



## Public Member Functions

- **MonitoringSystem (ScreenHardware \*screen, KeyboardHardware \*keyboard, SwitchHardware \*sw)**  
*Construct a new Monitoring System object.*
- **~MonitoringSystem ()**  
*Destroy the Monitoring System object.*
- int **getSelection ()**

- std::string **getName** ()  
*Get the Name object.*
- std::string **getNIF** ()  
*Get the NIF object.*
- std::string **getPassword** ()  
*Get the Password object.*
- std::string **getNameSelectedUser** ()  
*Get the Name Selected User object.*
- std::string **getNIFSelectedUser** ()  
*Get the NIF Selected User object.*
- std::string **getPasswordSelectedUser** ()  
*Get the Password Selected User object.*
- std::string **getPrivilegesSelectedUser** ()  
*Get the Privileges Selected User object.*
- std::string **getEmailSelectedUser** ()  
*Get the Email Selected User object.*
- int **getIdAlarm** ()  
*Get the Id Alarm object.*
- std::string **getTypeAlarm** ()  
*Get the Type Alarm object.*
- int **getIdSelectedAlarm** ()  
*Get the Id Selected Alarm object.*
- std::string **getTypeCamera** ()  
*Get the Type Camera object.*
- int **getIdCamera** ()  
*Get the Id Camera object.*
- int **getIdSelectedCamera** ()  
*Get the Id Selected Camera object.*
- void **initialScreen** ()  
*This method initializes the screen and keyboard.*
- void **exitScreen** ()  
*This method exits the screen.*
- void **loginScreen** ()  
*This method shows the login screen.*
- void **mainWindowAdmin** ()  
*This method shows the main menu for the admins.*
- void **mainWindowEmployee** ()  
*This method shows the main menu for the employees.*
- void **mainWindowGuest** ()  
*This method shows the main menu for the guests.*
- void **createUserScreen** ()  
*This method shows the message to create a new user.*
- void **deleteUserScreen** ()  
*This method shows the message to delete a user.*
- void **updateUserScreen** ()  
*This method shows the message to update a user.*
- void **displayUsersScreen** ()  
*This method shows the message to show the users.*
- void **displaySensorsScreen** ()  
*This method shows the message to show the sensors.*

- void **displayAlarmsScreen** ()  
*This method shows the message to show the alarms.*
- void **turnOnOffSystemScreen** ()  
*This method shows the message to turn on or off the system.*
- void **displayErrorScreen** ()  
*This method shows the message if an error occurs.*
- void **createAlarmScreen** ()  
*This method shows the message to create a new alarm.*
- void **deleteAlarmScreen** ()  
*This method shows the message to delete an alarm.*
- void **saveAlarmScreen** ()  
*This method shows the message to save an alarm.*
- void **displayCameraScreen** ()  
*Display the camera screen.*
- void **createCameraScreen** ()  
*Create a Camera Screen object.*
- void **deleteCameraScreen** ()  
*Delete a Camera Screen object.*
- void **turnOnOffCameraScreen** ()  
*Turn on or off a Camera Screen object.*
- void **saveCameraScreen** ()  
*Save a Camera Screen object.*
- void **cleanScreen** ()  
*Clean the screen.*
- void **saveUsersScreen** ()  
*Save the users screen.*

## Private Member Functions

- bool **inputCorrect** (int input, int max)  
*This method checks if the input is correct.*
- int **askInputInt** (int max)  
*This method asks the user for an int input.*
- std::string **askName** ()  
*This method asks the user for an input for the name.*
- std::string **askNIF** ()  
*This method asks the user for an input for the NIF.*
- std::string **askPassword** ()  
*This method asks the user for an input for the password.*
- std::string **askPrivileges** ()  
*This method asks the user for an input for the privileges.*
- std::string **askEmail** ()  
*This method asks the user for an input for the email.*
- void **selectUser** ()  
*This is the method to select a user.*
- void **shortSelectUser** ()  
*This is the method to select a short user.*
- void **askMainUser** ()  
*This is the method to ask the main user, the one that its going to loggin.*
- std::string **askTypeAlarm** ()

- `int askIdAlarm ()`  
*This is the method to ask the type of the alarm.*
- `std::string askTypeCamera ()`  
*This is the method to ask the id of the alarm.*
- `int askIdCamera ()`  
*This is the method to ask the type of the camera.*
- `std::string askTypeAlarm ()`  
*This is the method to ask the id of the camera.*

## Private Attributes

- `ScreenHardware * screen`  
*This is the pointer to the `ScreenHardware` object.*
- `KeyboardHardware * keyboard`  
*This is the pointer to the `KeyboardHardware` object.*
- `SwitchHardware * sw`  
*This is the pointer to the `SwitchHardware` object.*
- `int selection_`  
*This is the selection of the user.*
- `std::string name_`  
*This is the name of the user.*
- `std::string nif_`  
*This is the NIF of the user.*
- `std::string password_`  
*This is the password of the user.*
- `std::string nameSelectedUser_`  
*This is the name of the selected user.*
- `std::string nifSelectedUser_`  
*This is the password of the selected user.*
- `std::string passwordSelectedUser_`  
*This is the privileges of the selected user.*
- `std::string privilegesSelectedUser_`  
*This is the privileges of the selected user.*
- `std::string emailSelectedUser_`  
*This is the email of the selected user.*
- `int idAlarm_`  
*This attribute is the id of the alarm.*
- `std::string typeAlarm_`  
*This attribute is the type of the alarm.*
- `int idSelectedAlarm_`  
*This attribute is the id of the selected alarm.*
- `int idCamera_`  
*This attribute is the id of the camera.*
- `std::string typeCamera_`  
*This attribute is the type of the camera.*
- `int idSelectedCamera_`  
*This attribute is the id of the selected camera.*

### 4.18.1 Detailed Description

Definition at line 24 of file MonitoringSystem.h.

## 4.18.2 Constructor & Destructor Documentation

### 4.18.2.1 MonitoringSystem()

```
MonitoringSystem::MonitoringSystem (
    ScreenHardware * screen,
    KeyboardHardware * keyboard,
    SwitchHardware * sw ) [explicit]
```

Construct a new Monitoring System object.

#### Parameters

<i>screen</i>	
<i>keyboard</i>	
<i>sw</i>	

Definition at line 15 of file MonitoringSystem.cpp.

```
18     : screen(screen), keyboard(keyboard), sw(sw) {}
```

### 4.18.2.2 ~MonitoringSystem()

```
MonitoringSystem::~MonitoringSystem ()
```

Destroy the Monitoring System object.

Definition at line 20 of file MonitoringSystem.cpp.

```
20
21     delete screen;
22     delete keyboard;
23     delete sw;
24 }
```

References keyboard, screen, and sw.

## 4.18.3 Member Function Documentation

### 4.18.3.1 askEmail()

```
std::string MonitoringSystem::askEmail () [private]
```

This method asks the user for an input for the email.

**Returns**

```
std::string
```

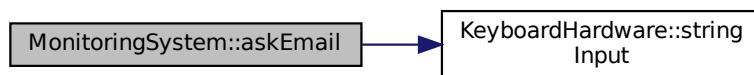
Definition at line 134 of file MonitoringSystem.cpp.

```
134
135     std::cout << "(EMAIL) ";
136     return keyboard->stringInput();
137 }
```

References keyboard, and KeyboardHardware::stringInput().

Referenced by selectUser().

Here is the call graph for this function:



Here is the caller graph for this function:

**4.18.3.2 askIdAlarm()**

```
int MonitoringSystem::askIdAlarm( ) [private]
```

This is the method to ask the id of the alarm.

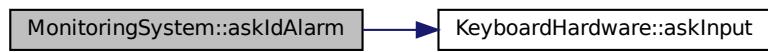
Definition at line 243 of file MonitoringSystem.cpp.

```
243
244     std::cout << "(ID ALARM) ";
245     return keyboard->askInput();
246 }
```

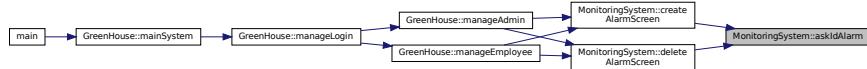
References KeyboardHardware::askInput(), and keyboard.

Referenced by createAlarmScreen(), and deleteAlarmScreen().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.3 askIdCamera()

```
int MonitoringSystem::askIdCamera () [private]
```

This is the method to ask the id of the camera.

**Returns**

```
int
```

Definition at line 287 of file MonitoringSystem.cpp.

```

287
288     // Pido con el keyboard un input que sera el id de la camara
289     std::cout << "(ID CAMERA) ";
290     return keyboard->askInput();
291 }
```

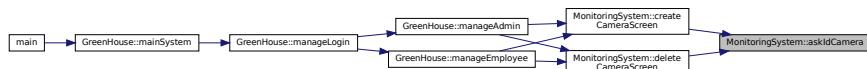
References KeyboardHardware::askInput(), and keyboard.

Referenced by createCameraScreen(), and deleteCameraScreen().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.4 askInputInt()

```
int MonitoringSystem::askInputInt (
    int max ) [private]
```

This method asks the user for an int input.

**Parameters**

<i>max</i>	
------------	--

**Returns**

int

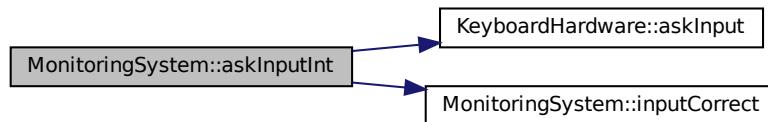
Definition at line 89 of file MonitoringSystem.cpp.

```
89     int input;
90
91     do {
92         input = keyboard->askInput();
93     } while (!inputCorrect(input, max));
94
95     return input;
96 }
```

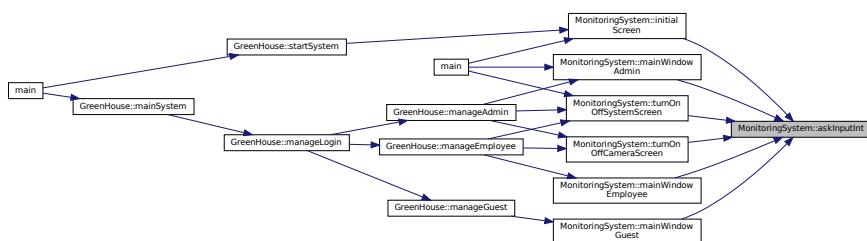
References KeyboardHardware::askInput(), inputCorrect(), and keyboard.

Referenced by initialScreen(), mainWindowAdmin(), mainWindowEmployee(), mainWindowGuest(), turnOnOff← CameraScreen(), and turnOnOffSystemScreen().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.5 askMainUser()

```
void MonitoringSystem::askMainUser ( ) [private]
```

This is the method to ask the main user, the one that its going to loggin.

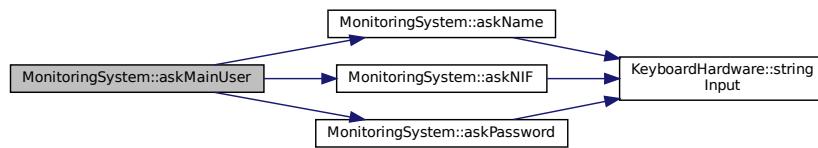
Definition at line 42 of file MonitoringSystem.cpp.

```
42
43     name_ = askName ();
44     password_ = askPassword ();
45     nif_ = askNIF ();
46 }
```

References askName(), askNIF(), askPassword(), name\_, nif\_, and password\_.

Referenced by loginScreen().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.6 askName()

```
std::string MonitoringSystem::askName ( ) [private]
```

This method asks the user for an input for the name.

**Returns**

```
std::string
```

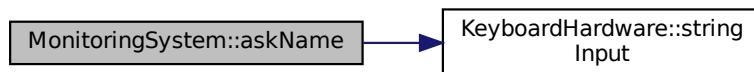
Definition at line 114 of file MonitoringSystem.cpp.

```
114     std::cout << "(NAME) ";
115     return keyboard->stringInput();
116 }
117 }
```

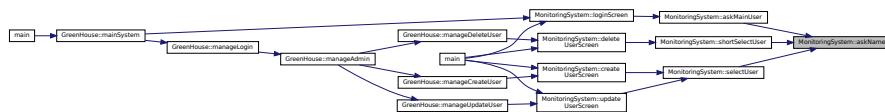
References keyboard, and KeyboardHardware::stringInput().

Referenced by askMainUser(), selectUser(), and shortSelectUser().

Here is the call graph for this function:



Here is the caller graph for this function:

**4.18.3.7 askNIF()**

```
std::string MonitoringSystem::askNIF( ) [private]
```

This method asks the user for an input for the NIF.

**Returns**

```
std::string
```

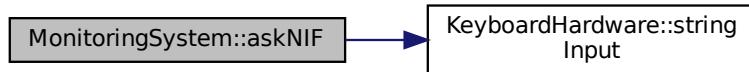
Definition at line 124 of file MonitoringSystem.cpp.

```
124
125     std::cout << "(NIF) ";
126     return keyboard->stringInput();
127 }
```

References keyboard, and KeyboardHardware::stringInput().

Referenced by askMainUser(), selectUser(), and shortSelectUser().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.8 askPassword()

```
std::string MonitoringSystem::askPassword ( ) [private]
```

This method asks the user for an input for the password.

**Returns**

std::string

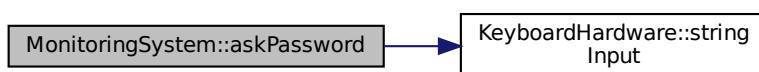
Definition at line 119 of file MonitoringSystem.cpp.

```
119 {
120     std::cout << "(PASSWORD) ";
121     return keyboard->stringInput ();
122 }
```

References `keyboard`, and `KeyboardHardware::stringInput()`.

Referenced by `askMainUser()`, `selectUser()`, and `shortSelectUser()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.9 askPrivileges()

```
std::string MonitoringSystem::askPrivileges () [private]
```

This method asks the user for an input for the privileges.

##### Returns

`std::string`

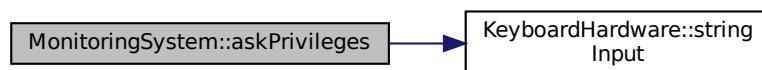
Definition at line 129 of file MonitoringSystem.cpp.

```
129
130     std::cout << "(PRIVILEGES) ";
131     return keyboard->stringInput ();
132 }
```

References `keyboard`, and `KeyboardHardware::stringInput()`.

Referenced by `selectUser()`.

Here is the call graph for this function:



Here is the caller graph for this function:



**4.18.3.10 askTypeAlarm()**

```
std::string MonitoringSystem::askTypeAlarm () [private]
```

This is the method to ask the type of the alarm.

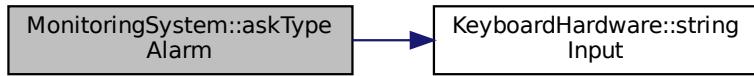
Definition at line 238 of file MonitoringSystem.cpp.

```
238     {
239     std::cout << "(TYPE ALARM) ";
240     return keyboard->stringInput();
241 }
```

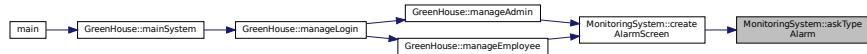
References keyboard, and KeyboardHardware::stringInput().

Referenced by createAlarmScreen().

Here is the call graph for this function:



Here is the caller graph for this function:

**4.18.3.11 askTypeCamera()**

```
std::string MonitoringSystem::askTypeCamera () [private]
```

This is the method to ask the type of the camera.

**Returns**

```
std::string
```

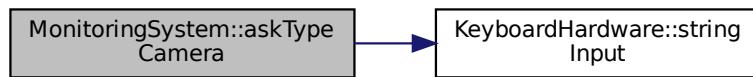
Definition at line 281 of file MonitoringSystem.cpp.

```
281      {
282      // Pido con el keyboard un input que sera el tipo de la camara
283      std::cout << "(TYPE CAMERA) ";
284      return keyboard->stringInput();
285 }
```

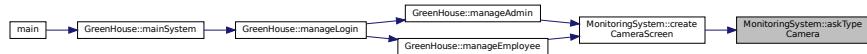
References keyboard, and KeyboardHardware::stringInput().

Referenced by createCameraScreen().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.12 cleanScreen()

```
void MonitoringSystem::cleanScreen( )
```

Clean the screen.

Definition at line 322 of file MonitoringSystem.cpp.

```
322      {
323      // Muestro de screen la cleanScreen
324      screen->cleanScreen();
325 }
```

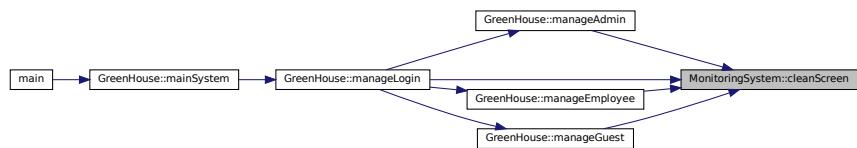
References ScreenHardware::cleanScreen(), and screen.

Referenced by GreenHouse::manageAdmin(), GreenHouse::manageEmployee(), GreenHouse::manageGuest(), and GreenHouse::manageLogin().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.13 createAlarmScreen()

```
void MonitoringSystem::createAlarmScreen ( )
```

This method shows the message to create a new alarm.

Definition at line 248 of file MonitoringSystem.cpp.

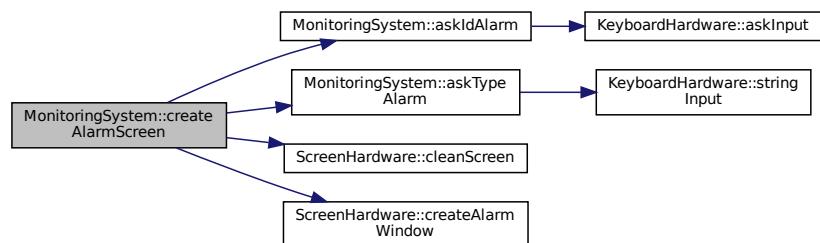
```

248
249     // Muestro de screen la createAlarmWindow
250     screen->cleanScreen();
251     screen->createAlarmWindow();
252     typeAlarm_ = askTypeAlarm();
253     idAlarm_ = askIdAlarm();
254 }
```

References askIdAlarm(), askTypeAlarm(), ScreenHardware::cleanScreen(), ScreenHardware::createAlarmWindow(), idAlarm\_, screen, and typeAlarm\_.

Referenced by GreenHouse::manageAdmin(), and GreenHouse::manageEmployee().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.14 createCameraScreen()

```
void MonitoringSystem::createCameraScreen ( )
```

Create a [Camera](#) Screen object.

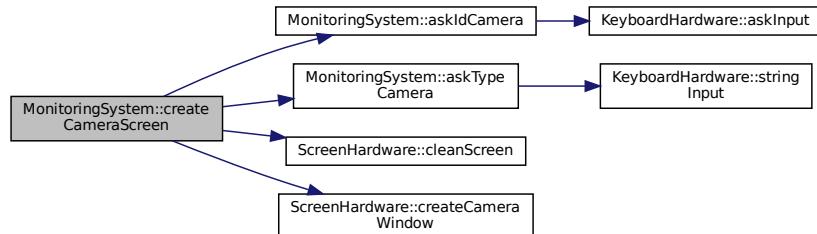
Definition at line 293 of file [MonitoringSystem.cpp](#).

```
293 {
294     // Muestro de screen la createCameraWindow
295     screen->cleanScreen();
296     screen->createCameraWindow();
297     idCamera_ = askIdCamera();
298     typeCamera_ = askTypeCamera();
299 }
```

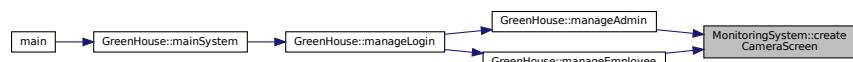
References [askIdCamera\(\)](#), [askTypeCamera\(\)](#), [ScreenHardware::cleanScreen\(\)](#), [ScreenHardware::createCameraWindow\(\)](#), [idCamera\\_](#), [screen](#), and [typeCamera\\_](#).

Referenced by [GreenHouse::manageAdmin\(\)](#), and [GreenHouse::manageEmployee\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.15 createUserScreen()

```
void MonitoringSystem::createUserScreen ( )
```

This method shows the message to create a new user.

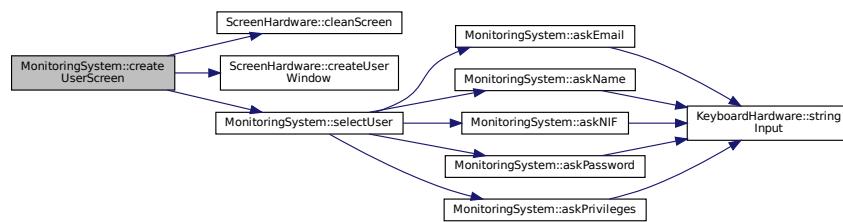
Definition at line 177 of file MonitoringSystem.cpp.

```
177      {
178      // Muestro de screen la createUserWindow, luego pido con el keyboard un input
179      // hasta que este entre los valores correctos
180      screen->cleanScreen();
181      screen->createUserWindow();
182      selectUser();
183 }
```

References ScreenHardware::cleanScreen(), ScreenHardware::createUserWindow(), screen, and selectUser().

Referenced by main(), and GreenHouse::manageCreateUser().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.16 deleteAlarmScreen()

```
void MonitoringSystem::deleteAlarmScreen ( )
```

This method shows the message to delete an alarm.

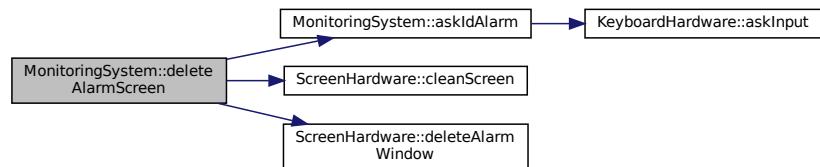
Definition at line 256 of file MonitoringSystem.cpp.

```
256      {
257      // Muestro de screen la deleteAlarmWindow
258      screen->cleanScreen();
259      screen->deleteAlarmWindow();
260      idSelectedAlarm_ = askIdAlarm();
261 }
```

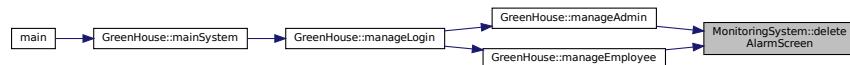
References askIdAlarm(), ScreenHardware::cleanScreen(), ScreenHardware::deleteAlarmWindow(), idSelectedAlarm\_, and screen.

Referenced by GreenHouse::manageAdmin(), and GreenHouse::manageEmployee().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.17 deleteCameraScreen()

```
void MonitoringSystem::deleteCameraScreen ( )
```

Delete a [Camera](#) Screen object.

Definition at line 301 of file `MonitoringSystem.cpp`.

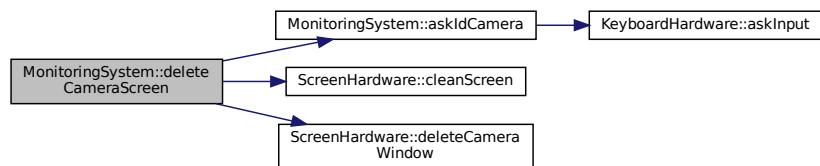
```

301  {
302  // Muestro de screen la deleteCameraWindow
303  screen->cleanScreen();
304  screen->deleteCameraWindow();
305  idSelectedCamera_ = askIdCamera();
306 }
```

References `askIdCamera()`, `ScreenHardware::cleanScreen()`, `ScreenHardware::deleteCameraWindow()`, `idSelectedCamera_`, and `screen`.

Referenced by GreenHouse::manageAdmin(), and GreenHouse::manageEmployee().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.18 deleteUserScreen()

```
void MonitoringSystem::deleteUserScreen ( )
```

This method shows the message to delete a user.

Definition at line 185 of file MonitoringSystem.cpp.

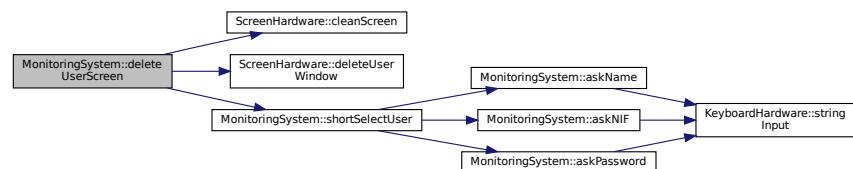
```

185   {
186     // Muestro de screen la deleteUserWindow, luego pido con el keyboard un input
187     // hasta que este entre los valores correctos
188     screen->cleanScreen();
189     screen->deleteUserWindow();
190     shortSelectUser();
191 }
```

References ScreenHardware::cleanScreen(), ScreenHardware::deleteUserWindow(), screen, and shortSelectUser().

Referenced by main(), and GreenHouse::manageDeleteUser().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.19 displayAlarmsScreen()

```
void MonitoringSystem::displayAlarmsScreen ( )
```

This method shows the message to show the alarms.

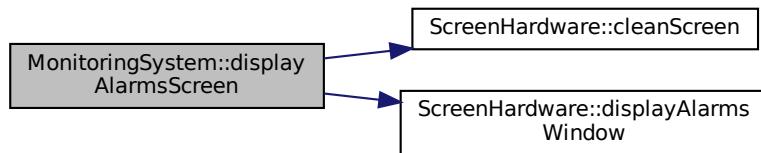
Definition at line 215 of file MonitoringSystem.cpp.

```
215   {
216     // Muestro de screen la displayAlarmsWindow
217     screen->cleanScreen();
218     screen->displayAlarmsWindow();
219     // keyboard->stringInput();
220 }
```

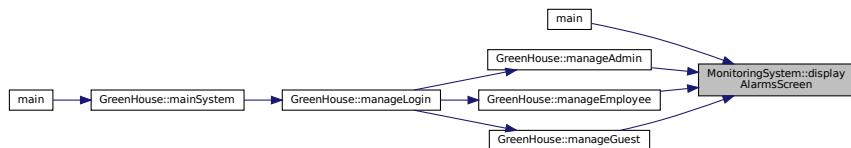
References ScreenHardware::cleanScreen(), ScreenHardware::displayAlarmsWindow(), and screen.

Referenced by main(), GreenHouse::manageAdmin(), GreenHouse::manageEmployee(), and GreenHouse::manageGuest().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.20 displayCameraScreen()

```
void MonitoringSystem::displayCameraScreen ( )
```

Display the camera screen.

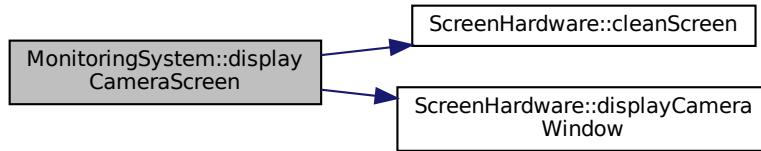
Definition at line 275 of file MonitoringSystem.cpp.

```
275   {
276     // Muestro de screen la displayCameraWindow
277     screen->cleanScreen();
278     screen->displayCameraWindow();
279 }
```

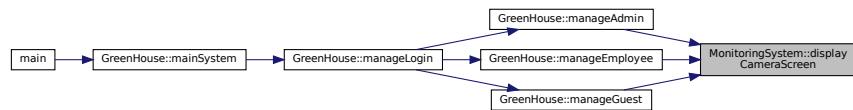
References ScreenHardware::cleanScreen(), ScreenHardware::displayCameraWindow(), and screen.

Referenced by GreenHouse::manageAdmin(), GreenHouse::manageEmployee(), and GreenHouse::manageGuest().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.21 displayErrorScreen()

```
void MonitoringSystem::displayErrorScreen ( )
```

This method shows the message if an error occurs.

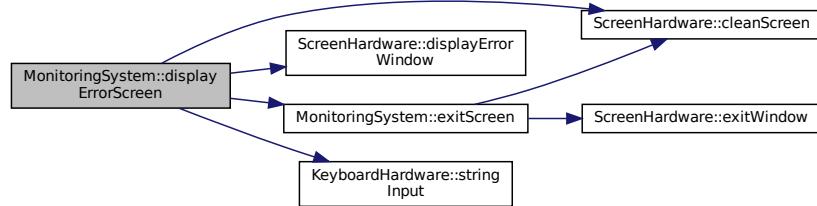
Definition at line 230 of file MonitoringSystem.cpp.

```
230
231     // Muestro de screen la displayErrorWindow
232     screen->cleanScreen();
233     screen->displayErrorWindow();
234     keyboard->stringInput();
235     exitScreen();
236 }
```

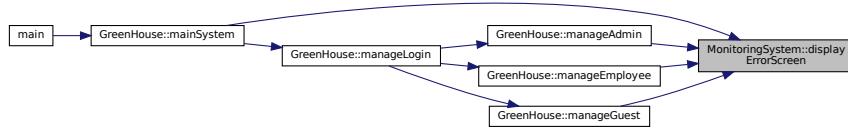
References ScreenHardware::cleanScreen(), ScreenHardware::displayErrorWindow(), exitScreen(), keyboard, screen, and KeyboardHardware::stringInput().

Referenced by GreenHouse::mainSystem(), GreenHouse::manageAdmin(), GreenHouse::manageEmployee(), and GreenHouse::manageGuest().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.22 `displaySensorsScreen()`

```
void MonitoringSystem::displaySensorsScreen ( )
```

This method shows the message to show the sensors.

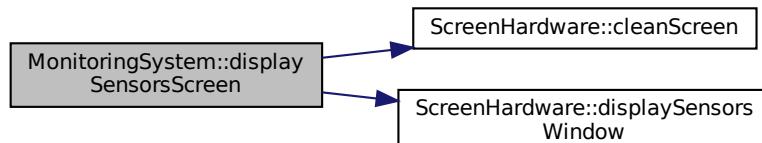
Definition at line 208 of file `MonitoringSystem.cpp`.

```
208
209     // Muestro de screen la displaySensorsWindow
210     screen->cleanScreen();
211     screen->displaySensorsWindow();
212     // keyboard->stringInput();
213 }
```

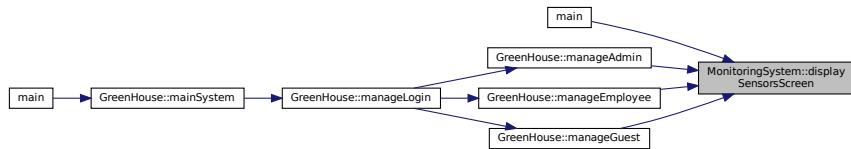
References `ScreenHardware::cleanScreen()`, `ScreenHardware::displaySensorsWindow()`, and `screen`.

Referenced by `main()`, `GreenHouse::manageAdmin()`, `GreenHouse::manageEmployee()`, and `GreenHouse::manageGuest()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.23 displayUsersScreen()

```
void MonitoringSystem::displayUsersScreen ( )
```

This method shows the message to show the users.

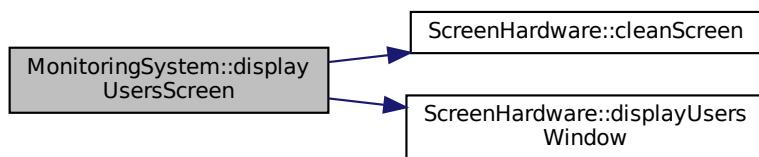
Definition at line 201 of file MonitoringSystem.cpp.

```
201
202     // Muestro de screen la displayUsersWindow
203     screen->cleanScreen();
204     screen->displayUsersWindow();
205     // keyboard->stringInput();
206 }
```

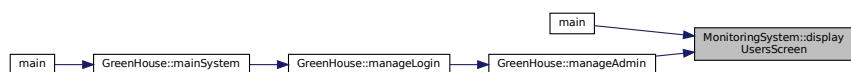
References ScreenHardware::cleanScreen(), ScreenHardware::displayUsersWindow(), and screen.

Referenced by main(), and GreenHouse::manageAdmin().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.24 exitScreen()

```
void MonitoringSystem::exitScreen ( )
```

This method exits the screen.

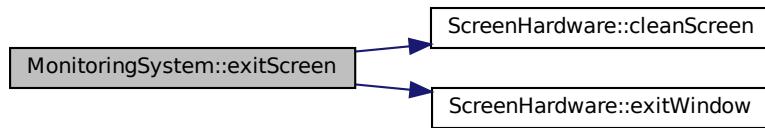
Definition at line 108 of file MonitoringSystem.cpp.

```
108
109 // Borrar terminal y mostrar el exitWindow
110 screen->cleanScreen();
111 screen->exitWindow();
112 }
```

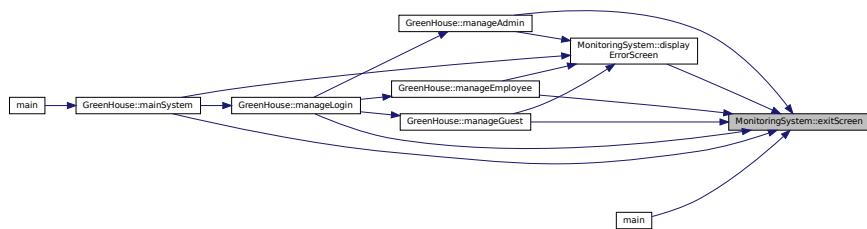
References ScreenHardware::cleanScreen(), ScreenHardware::exitWindow(), and screen.

Referenced by displayErrorScreen(), main(), GreenHouse::mainSystem(), GreenHouse::manageAdmin(), GreenHouse::manageEmployee(), GreenHouse::manageGuest(), and GreenHouse::manageLogin().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.25 getEmailSelectedUser()

```
std::string MonitoringSystem::getEmailSelectedUser ( )
```

Get the Email Selected [User](#) object.

**Returns**

```
std::string
```

Definition at line 74 of file MonitoringSystem.cpp.

```
74
75   return emailSelectedUser_;
76 }
```

References emailSelectedUser\_.

Referenced by GreenHouse::manageCreateUser(), and GreenHouse::manageUpdateUser().

Here is the caller graph for this function:

**4.18.3.26 getIdAlarm()**

```
int MonitoringSystem::getIdAlarm ( )
```

Get the Id Alarm object.

**Returns**

```
int
```

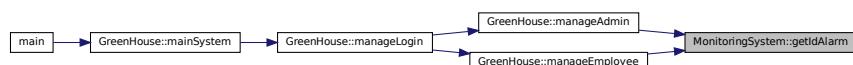
Definition at line 50 of file MonitoringSystem.cpp.

```
50 { return idAlarm_; }
```

References idAlarm\_.

Referenced by GreenHouse::manageAdmin(), and GreenHouse::manageEmployee().

Here is the caller graph for this function:



#### 4.18.3.27 getIdCamera()

```
int MonitoringSystem::getIdCamera ( )
```

Get the Id [Camera](#) object.

Returns

int

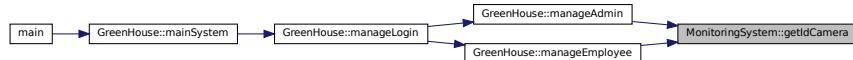
Definition at line 271 of file [MonitoringSystem.cpp](#).

```
271 { return idCamera_; }
```

References [idCamera\\_](#).

Referenced by [GreenHouse::manageAdmin\(\)](#), and [GreenHouse::manageEmployee\(\)](#).

Here is the caller graph for this function:



#### 4.18.3.28 getIdSelectedAlarm()

```
int MonitoringSystem::getIdSelectedAlarm ( )
```

Get the Id Selected Alarm object.

Returns

int

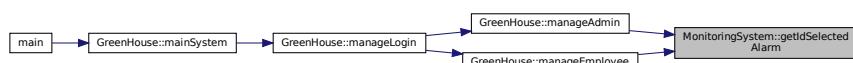
Definition at line 52 of file [MonitoringSystem.cpp](#).

```
52 { return idSelectedAlarm_; }
```

References [idSelectedAlarm\\_](#).

Referenced by [GreenHouse::manageAdmin\(\)](#), and [GreenHouse::manageEmployee\(\)](#).

Here is the caller graph for this function:



#### 4.18.3.29 getIdSelectedCamera()

```
int MonitoringSystem::getIdSelectedCamera ( )
```

Get the Id Selected [Camera](#) object.

##### Returns

int

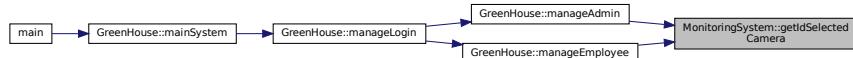
Definition at line 273 of file [MonitoringSystem.cpp](#).

```
273 { return idSelectedCamera_; }
```

References [idSelectedCamera\\_](#).

Referenced by [GreenHouse::manageAdmin\(\)](#), and [GreenHouse::manageEmployee\(\)](#).

Here is the caller graph for this function:



#### 4.18.3.30 getName()

```
std::string MonitoringSystem::getName ( )
```

Get the Name object.

##### Returns

std::string

Definition at line 54 of file [MonitoringSystem.cpp](#).

```
54 { return name_; }
```

References [name\\_](#).

Referenced by [GreenHouse::manageLogin\(\)](#).

Here is the caller graph for this function:



#### 4.18.3.31 getNameSelectedUser()

```
std::string MonitoringSystem::getNameSelectedUser ( )
```

Get the Name Selected [User](#) object.

##### Returns

`std::string`

Definition at line 60 of file `MonitoringSystem.cpp`.

```
60
61     return nameSelectedUser_;
62 }
```

References `nameSelectedUser_`.

Referenced by `GreenHouse::manageCreateUser()`, `GreenHouse::manageDeleteUser()`, and `GreenHouse::manageUpdateUser()`.

Here is the caller graph for this function:



#### 4.18.3.32 getNIF()

```
std::string MonitoringSystem::getNIF ( )
```

Get the NIF object.

##### Returns

`std::string`

Definition at line 56 of file `MonitoringSystem.cpp`.

```
56 { return nif_; }
```

References `nif_`.

Referenced by `GreenHouse::manageLogin()`.

Here is the caller graph for this function:



#### 4.18.3.33 getNIFSelectedUser()

```
std::string MonitoringSystem::getNIFSelectedUser ( )
```

Get the NIF Selected [User](#) object.

##### Returns

`std::string`

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

```
64 { return nifSelectedUser_; }
```

References `nifSelectedUser_`.

Referenced by `GreenHouse::manageCreateUser()`, `GreenHouse::manageDeleteUser()`, and `GreenHouse::manageUpdateUser()`.

Here is the caller graph for this function:



#### 4.18.3.34 getPassword()

```
std::string MonitoringSystem::getPassword ( )
```

Get the Password object.

##### Returns

`std::string`

Definition at line 58 of file `MonitoringSystem.cpp`.

```
58 { return password_; }
```

References `password_`.

Referenced by `GreenHouse::manageLogin()`.

Here is the caller graph for this function:



#### 4.18.3.35 getPasswordSelectedUser()

```
std::string MonitoringSystem::getPasswordSelectedUser ( )
```

Get the Password Selected User object.

##### Returns

```
std::string
```

Definition at line 66 of file MonitoringSystem.cpp.

```
66
67     return passwordSelectedUser_;
68 }
```

References passwordSelectedUser\_.

Referenced by GreenHouse::manageCreateUser(), GreenHouse::manageDeleteUser(), and GreenHouse::manageUpdateUser().

Here is the caller graph for this function:



#### 4.18.3.36 getPrivilegesSelectedUser()

```
std::string MonitoringSystem::getPrivilegesSelectedUser ( )
```

Get the Privileges Selected User object.

##### Returns

```
std::string
```

Definition at line 70 of file MonitoringSystem.cpp.

```
70
71     return privilegesSelectedUser_;
72 }
```

References privilegesSelectedUser\_.

Referenced by GreenHouse::manageCreateUser(), and GreenHouse::manageUpdateUser().

Here is the caller graph for this function:



**4.18.3.37 getSelection()**

```
int MonitoringSystem::getSelection ( )
```

Get the Selection object.

**Returns**

int

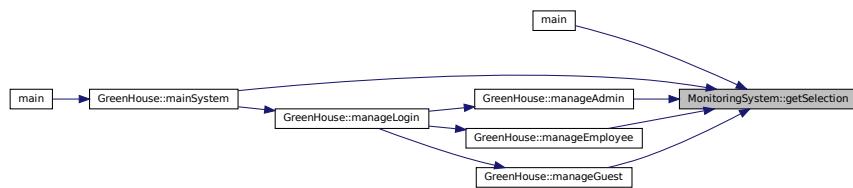
Definition at line 26 of file MonitoringSystem.cpp.

```
26 { return selection_; }
```

References selection\_.

Referenced by main(), GreenHouse::mainSystem(), GreenHouse::manageAdmin(), GreenHouse::manageEmployee(), and GreenHouse::manageGuest().

Here is the caller graph for this function:

**4.18.3.38 getTypeAlarm()**

```
std::string MonitoringSystem::getTypeAlarm ( )
```

Get the Type Alarm object.

**Returns**

`std::string`

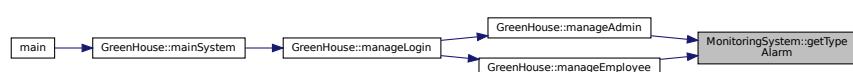
Definition at line 48 of file MonitoringSystem.cpp.

```
48 { return typeAlarm_; }
```

References typeAlarm\_.

Referenced by GreenHouse::manageAdmin(), and GreenHouse::manageEmployee().

Here is the caller graph for this function:



#### 4.18.3.39 getTypeCamera()

```
std::string MonitoringSystem::getTypeCamera ( )
```

Get the Type Camera object.

##### Returns

```
std::string
```

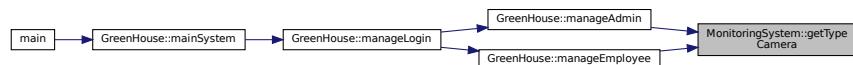
Definition at line 269 of file MonitoringSystem.cpp.

```
269 { return typeCamera_; }
```

References typeCamera\_.

Referenced by GreenHouse::manageAdmin(), and GreenHouse::manageEmployee().

Here is the caller graph for this function:



#### 4.18.3.40 initialScreen()

```
void MonitoringSystem::initialScreen ( )
```

This method initializes the screen and keyboard.

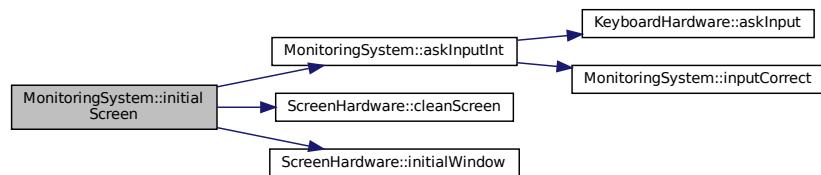
Definition at line 99 of file MonitoringSystem.cpp.

```
99
100   screen->cleanScreen();
101   int options = MAIN_MENU_OPTIONS;
102   // Muestro de screen la initialWindow, luego pido con el keyboard un input
103   // hasta que este entre los valores correctos
104   screen->initialWindow();
105   selection_ = askInputInt(options);
106 }
```

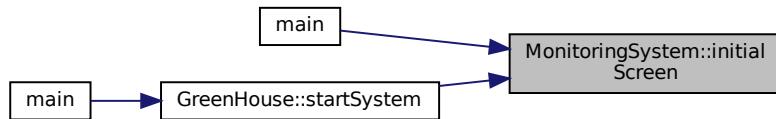
References askInputInt(), ScreenHardware::cleanScreen(), ScreenHardware::initialWindow(), MAIN\_MENU\_OPTIONS, screen, and selection\_.

Referenced by main(), and GreenHouse::startSystem().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.41 inputCorrect()

```
bool MonitoringSystem::inputCorrect (
    int input,
    int max ) [private]
```

This method checks if the input is correct.

##### Parameters

<i>input</i>	
<i>max</i>	

##### Returns

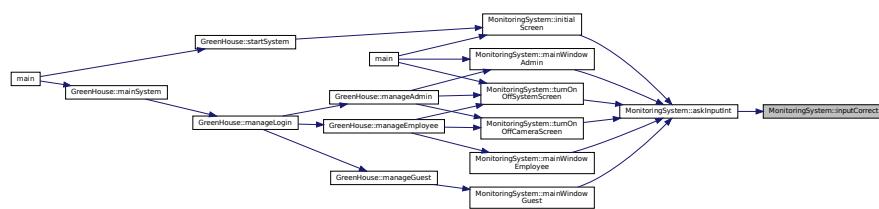
true  
false

Definition at line 78 of file MonitoringSystem.cpp.

```
78
79     bool correct = input >= 1 && input <= max;
80     if (!correct) {
81         cout << "Invalid input. Please enter an integer that corresponds to one of "
82             << "the options"
83             << endl;
84     }
85 // Input debe de estar entre 1 y max
86     return correct;
87 }
```

Referenced by askInputInt().

Here is the caller graph for this function:



#### 4.18.3.42 loginScreen()

```
void MonitoringSystem::loginScreen ( )
```

This method shows the login screen.

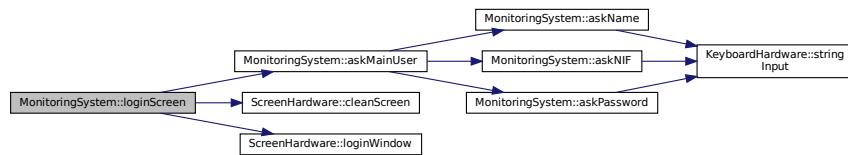
Definition at line 139 of file MonitoringSystem.cpp.

```
139
140     // Muestro de screen la loginWindow, luego pido con el keyboard un input hasta
141     // que este entre los valores correctos
142     screen->cleanScreen();
143     screen->loginWindow();
144     askMainUser();
145     // std::cout << name_ << " " << password_ << " " << nif_ << endl;
146 }
```

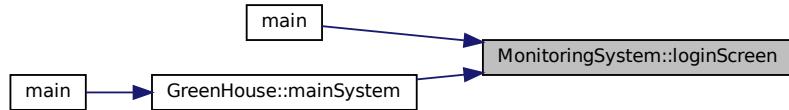
References askMainUser(), ScreenHardware::cleanScreen(), ScreenHardware::loginWindow(), and screen.

Referenced by main(), and GreenHouse::mainSystem().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.43 mainWindowAdmin()

```
void MonitoringSystem::mainWindowAdmin ( )
```

This method shows the main menu for the admins.

Definition at line 148 of file MonitoringSystem.cpp.

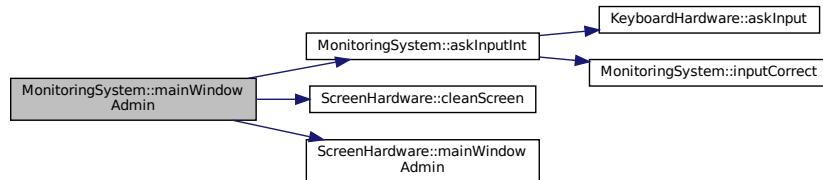
```
148
149     // Muestro de screen la mainWindowAdmin, luego pido con el keyboard un input
150     // hasta que este entre los valores correctos
151     screen->cleanScreen();
152     int options = ADMIN_MENU_OPTIONS;
153     screen->mainWindowAdmin();
154     selection_ = askInputInt(options);
```

```
155 }
```

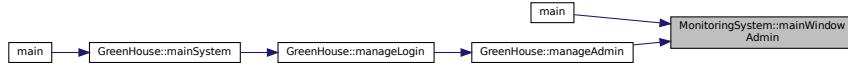
References ADMIN\_MENU\_OPTIONS, askInputInt(), ScreenHardware::cleanScreen(), ScreenHardware::mainWindowAdmin(), screen, and selection\_.

Referenced by main(), and GreenHouse::manageAdmin().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.44 mainWindowEmployee()

```
void MonitoringSystem::mainWindowEmployee ( )
```

This method shows the main menu for the employees.

Definition at line 157 of file MonitoringSystem.cpp.

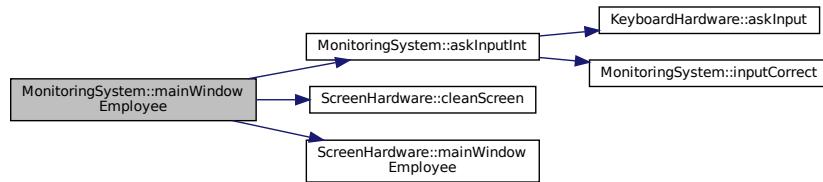
```

157   {
158     // Muestro de screen la mainWindowEmployee, luego pido con el keyboard un
159     // input hasta que este entre los valores correctos
160     screen->cleanScreen();
161     int options = EMPLOYEE_MENU_OPTIONS;
162     screen->mainWindowEmployee();
163     selection_ = askInputInt(options);
164 }
```

References askInputInt(), ScreenHardware::cleanScreen(), EMPLOYEE\_MENU\_OPTIONS, ScreenHardware::mainWindowEmployee(), screen, and selection\_.

Referenced by GreenHouse::manageEmployee().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.45 mainWindowGuest()

```
void MonitoringSystem::mainWindowGuest ( )
```

This method shows the main menu for the guests.

Definition at line 166 of file `MonitoringSystem.cpp`.

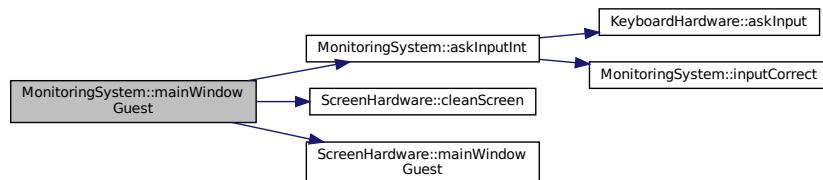
```

166      {
167      // Muestro de screen la mainWindowGuest, luego pido con el keyboard un input
168      // hasta que este entre los valores correctos
169      screen->cleanScreen();
170      int options = GUEST_MENU_OPTIONS;
171      screen->mainWindowGuest ();
172      selection_ = askInputInt(options);
173  }
```

References `askInputInt()`, `ScreenHardware::cleanScreen()`, `GUEST_MENU_OPTIONS`, `ScreenHardware::mainWindowGuest()`, `screen`, and `selection_`.

Referenced by `GreenHouse::manageGuest()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.46 saveAlarmScreen()

```
void MonitoringSystem::saveAlarmScreen ( )
```

This method shows the message to save an alarm.

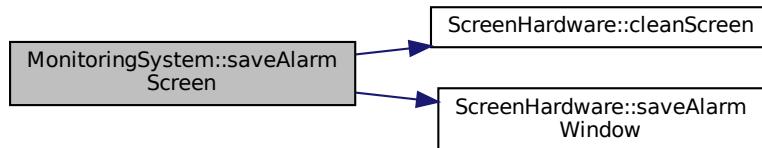
Definition at line 263 of file MonitoringSystem.cpp.

```
263 {  
264     // Muestro de screen la saveAlarmWindow  
265     screen->cleanScreen();  
266     screen->saveAlarmWindow();  
267 }
```

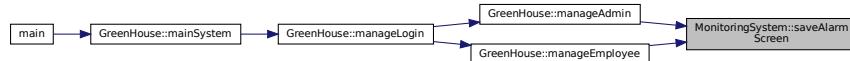
References ScreenHardware::cleanScreen(), ScreenHardware::saveAlarmWindow(), and screen.

Referenced by GreenHouse::manageAdmin(), and GreenHouse::manageEmployee().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.47 saveCameraScreen()

```
void MonitoringSystem::saveCameraScreen ( )
```

Save a [Camera](#) Screen object.

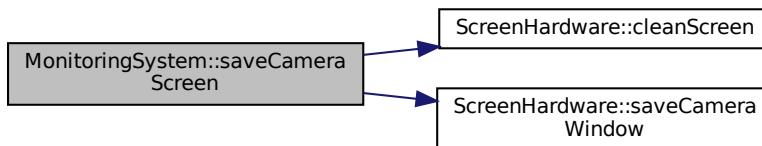
Definition at line 316 of file MonitoringSystem.cpp.

```
316      {
317      // Muestro de screen la saveCameraWindow
318      screen->cleanScreen();
319      screen->saveCameraWindow();
320  }
```

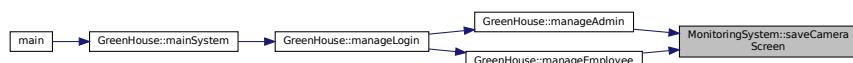
References ScreenHardware::cleanScreen(), ScreenHardware::saveCameraWindow(), and screen.

Referenced by GreenHouse::manageAdmin(), and GreenHouse::manageEmployee().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.48 saveUsersScreen()

```
void MonitoringSystem::saveUsersScreen ( )
```

Save the users screen.

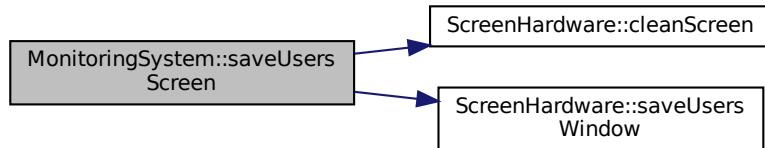
Definition at line 327 of file MonitoringSystem.cpp.

```
327      {
328      // Muestro de screen la saveUsersWindow
329      screen->cleanScreen();
330      screen->saveUsersWindow();
331  }
```

References ScreenHardware::cleanScreen(), ScreenHardware::saveUsersWindow(), and screen.

Referenced by GreenHouse::manageAdmin().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.49 selectUser()

```
void MonitoringSystem::selectUser () [private]
```

This is the method to select a user.

In this selection you have to introduce all the parameters of the user.

Definition at line 28 of file MonitoringSystem.cpp.

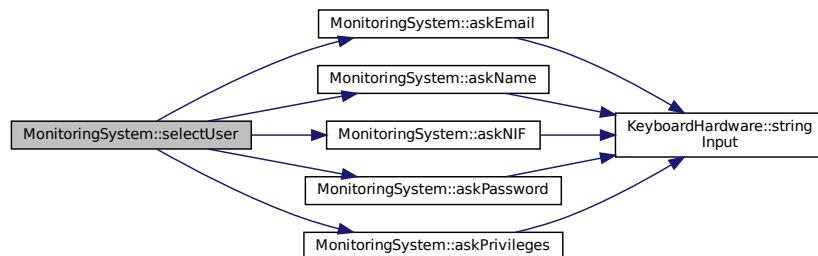
```

28
29     nameSelectedUser_ = askName ();
30     passwordSelectedUser_ = askPassword ();
31     nifSelectedUser_ = askNIF ();
32     privilegesSelectedUser_ = askPrivileges ();
33     emailSelectedUser_ = askEmail ();
34 }
```

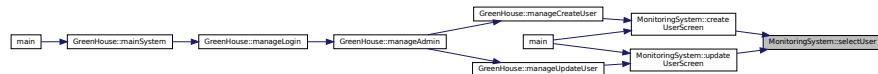
References askEmail(), askName(), askNIF(), askPassword(), askPrivileges(), emailSelectedUser\_, nameSelectedUser\_, nifSelectedUser\_, passwordSelectedUser\_, and privilegesSelectedUser\_.

Referenced by createUserScreen(), and updateUserScreen().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.50 shortSelectUser()

```
void MonitoringSystem::shortSelectUser () [private]
```

This is the method to select a short user.

In this selection you have to introduce the name, the password and the NIF of the user.

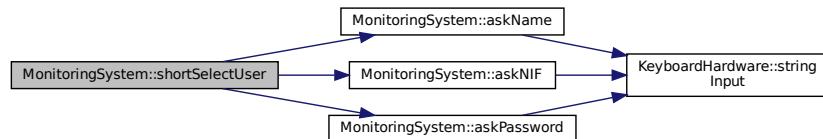
Definition at line 36 of file MonitoringSystem.cpp.

```
36 {
37     nameSelectedUser_ = askName ();
38     passwordSelectedUser_ = askPassword ();
39     nifSelectedUser_ = askNIF ();
40 }
```

References askName(), askNIF(), askPassword(), nameSelectedUser\_, nifSelectedUser\_, and passwordSelectedUser\_.

Referenced by deleteUserScreen().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.51 turnOnOffCameraScreen()

```
void MonitoringSystem::turnOnOffCameraScreen ( )
```

Turn on or off a Camera Screen object.

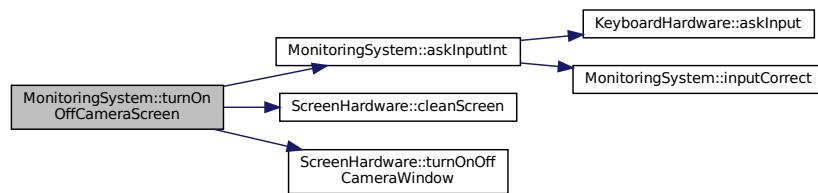
Definition at line 308 of file MonitoringSystem.cpp.

```
308
309 // Muestro de screen la turnOnOffCameraWindow
310 screen->cleanScreen();
311 int options = 2;
312 screen->turnOnOffCameraWindow();
313 selection_ = askInputInt(options);
314 }
```

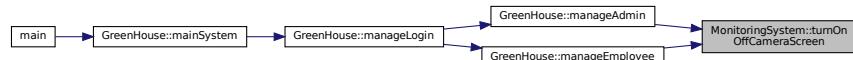
References askInputInt(), ScreenHardware::cleanScreen(), screen, selection\_, and ScreenHardware::turnOnOffCameraWindow().

Referenced by GreenHouse::manageAdmin(), and GreenHouse::manageEmployee().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.52 turnOnOffSystemScreen()

```
void MonitoringSystem::turnOnOffSystemScreen ( )
```

This method shows the message to turn on or off the system.

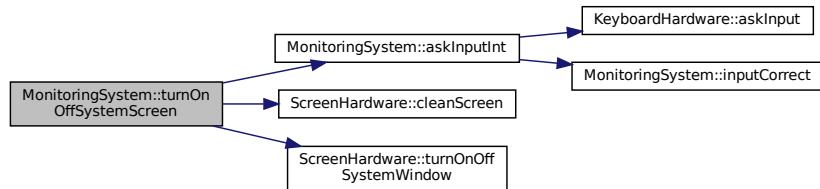
Definition at line 222 of file MonitoringSystem.cpp.

```
222
223 // Muestro de screen la turnOnOffSystemWindow
224 screen->cleanScreen();
225 int options = 2;
226 screen->turnOnOffSystemWindow();
227 selection_ = askInputInt(options);
228 }
```

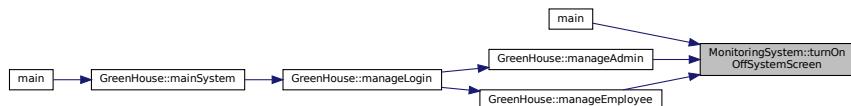
References askInputInt(), ScreenHardware::cleanScreen(), screen, selection\_, and ScreenHardware::turnOnOffSystemWindow().

Referenced by main(), GreenHouse::manageAdmin(), and GreenHouse::manageEmployee().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.3.53 updateUserScreen()

```
void MonitoringSystem::updateUserScreen ( )
```

This method shows the message to update a user.

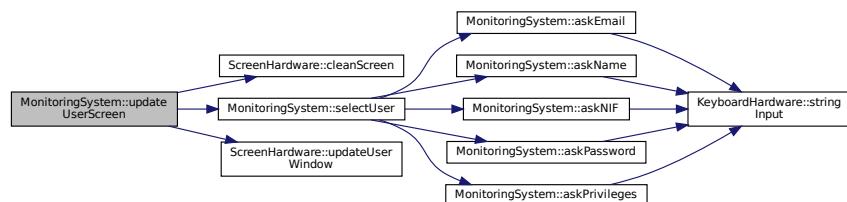
Definition at line 193 of file MonitoringSystem.cpp.

```
193  {
194  // Muestro de screen la updateUserWindow, luego pido con el keyboard un input
195  // hasta que este entre los valores correctos
196  screen->cleanScreen();
197  screen->updateUserWindow();
198  selectUser();
199 }
```

References ScreenHardware::cleanScreen(), screen, selectUser(), and ScreenHardware::updateUserWindow().

Referenced by main(), and GreenHouse::manageUpdateUser().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.18.4 Member Data Documentation

##### 4.18.4.1 emailSelectedUser\_

```
std::string MonitoringSystem::emailSelectedUser_ [private]
```

This is the email of the selected user.

Definition at line 414 of file MonitoringSystem.h.

Referenced by getEmailSelectedUser(), and selectUser().

##### 4.18.4.2 idAlarm\_

```
int MonitoringSystem::idAlarm_ [private]
```

This attribute is the id of the alarm.

Definition at line 430 of file MonitoringSystem.h.

Referenced by createAlarmScreen(), and getIdAlarm().

##### 4.18.4.3 idCamera\_

```
int MonitoringSystem::idCamera_ [private]
```

This attribute is the id of the camera.

Definition at line 457 of file MonitoringSystem.h.

Referenced by createCameraScreen(), and getIdCamera().

#### 4.18.4.4 **idSelectedAlarm\_**

```
int MonitoringSystem::idSelectedAlarm_ [private]
```

This attribute is the id of the selected alarm.

Definition at line 438 of file MonitoringSystem.h.

Referenced by deleteAlarmScreen(), and getIdSelectedAlarm().

#### 4.18.4.5 **idSelectedCamera\_**

```
int MonitoringSystem::idSelectedCamera_ [private]
```

This attribute is the id of the selected camera.

Definition at line 467 of file MonitoringSystem.h.

Referenced by deleteCameraScreen(), and getIdSelectedCamera().

#### 4.18.4.6 **keyboard**

```
KeyboardHardware* MonitoringSystem::keyboard [private]
```

This is the pointer to the [KeyboardHardware](#) object.

Definition at line 287 of file MonitoringSystem.h.

Referenced by askEmail(), askIdAlarm(), askIdCamera(), askInputInt(), askName(), askNIF(), askPassword(), askPrivileges(), askTypeAlarm(), askTypeCamera(), displayErrorScreen(), and ~MonitoringSystem().

#### 4.18.4.7 **name\_**

```
std::string MonitoringSystem::name_ [private]
```

This is the name of the user.

Definition at line 352 of file MonitoringSystem.h.

Referenced by askMainUser(), and getName().

**4.18.4.8 nameSelectedUser\_**

```
std::string MonitoringSystem::nameSelectedUser_ [private]
```

This is the name of the selected user.

This is the NIF of the selected user.

Definition at line 394 of file MonitoringSystem.h.

Referenced by getNameSelectedUser(), selectUser(), and shortSelectUser().

**4.18.4.9 nif\_**

```
std::string MonitoringSystem::nif_ [private]
```

This is the NIF of the user.

Definition at line 357 of file MonitoringSystem.h.

Referenced by askMainUser(), and getNIF().

**4.18.4.10 nifSelectedUser\_**

```
std::string MonitoringSystem::nifSelectedUser_ [private]
```

This is the password of the selected user.

Definition at line 399 of file MonitoringSystem.h.

Referenced by getNIFSelectedUser(), selectUser(), and shortSelectUser().

**4.18.4.11 password\_**

```
std::string MonitoringSystem::password_ [private]
```

This is the password of the user.

Definition at line 362 of file MonitoringSystem.h.

Referenced by askMainUser(), and getPassword().

#### 4.18.4.12 **passwordSelectedUser\_**

```
std::string MonitoringSystem::passwordSelectedUser_ [private]
```

This is the privileges of the selected user.

Definition at line 404 of file MonitoringSystem.h.

Referenced by getPasswordSelectedUser(), selectUser(), and shortSelectUser().

#### 4.18.4.13 **privilegesSelectedUser\_**

```
std::string MonitoringSystem::privilegesSelectedUser_ [private]
```

This is the privileges of the selected user.

Definition at line 409 of file MonitoringSystem.h.

Referenced by getPrivilegesSelectedUser(), and selectUser().

#### 4.18.4.14 **screen**

```
ScreenHardware* MonitoringSystem::screen [private]
```

This is the pointer to the [ScreenHardware](#) object.

Definition at line 282 of file MonitoringSystem.h.

Referenced by cleanScreen(), createAlarmScreen(), createCameraScreen(), createUserScreen(), deleteAlarmScreen(), deleteCameraScreen(), deleteUserScreen(), displayAlarmsScreen(), displayCameraScreen(), displayErrorScreen(), displaySensorsScreen(), displayUsersScreen(), exitScreen(), initialScreen(), loginScreen(), mainWindowAdmin(), mainWindowEmployee(), mainWindowGuest(), saveAlarmScreen(), saveCameraScreen(), saveUsersScreen(), turnOnOffCameraScreen(), turnOnOffSystemScreen(), updateUserScreen(), and ~MonitoringSystem().

#### 4.18.4.15 **selection\_**

```
int MonitoringSystem::selection_ [private]
```

This is the selection of the user.

Definition at line 347 of file MonitoringSystem.h.

Referenced by getSelection(), initialScreen(), mainWindowAdmin(), mainWindowEmployee(), mainWindowGuest(), turnOnOffCameraScreen(), and turnOnOffSystemScreen().

#### 4.18.4.16 **sw**

```
SwitchHardware* MonitoringSystem::sw [private]
```

This is the pointer to the [SwitchHardware](#) object.

Definition at line 292 of file [MonitoringSystem.h](#).

Referenced by [~MonitoringSystem\(\)](#).

#### 4.18.4.17 **typeAlarm\_**

```
std::string MonitoringSystem::typeAlarm_ [private]
```

This attribute is the type of the alarm.

Definition at line 434 of file [MonitoringSystem.h](#).

Referenced by [createAlarmScreen\(\)](#), and [getTypeAlarm\(\)](#).

#### 4.18.4.18 **typeCamera\_**

```
std::string MonitoringSystem::typeCamera_ [private]
```

This attribute is the type of the camera.

Definition at line 462 of file [MonitoringSystem.h](#).

Referenced by [createCameraScreen\(\)](#), and [getTypeCamera\(\)](#).

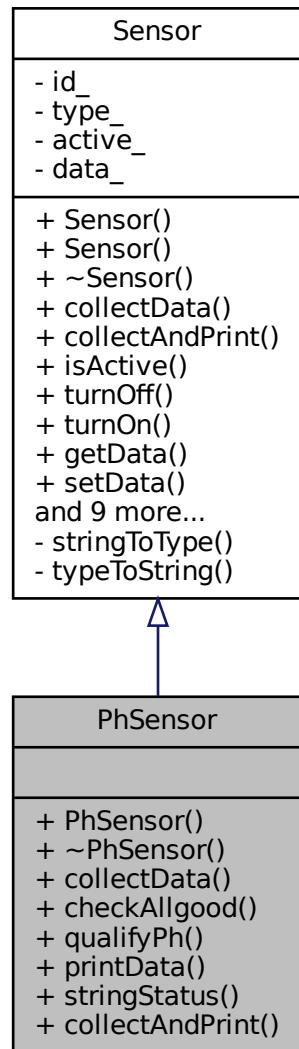
The documentation for this class was generated from the following files:

- src/[MonitoringSystem.h](#)
- src/[MonitoringSystem.cpp](#)

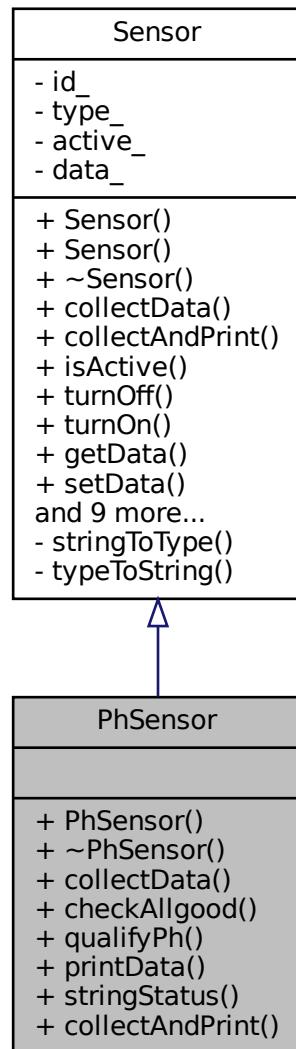
## 4.19 PhSensor Class Reference

```
#include <PhSensor.h>
```

Inheritance diagram for PhSensor:



Collaboration diagram for PhSensor:



## Public Member Functions

- **PhSensor** (int id, bool active)
 

*Construct a new Ph Sensor object.*
- **~PhSensor ()** override
 

*Destroy the Ph Sensor object.*
- void **collectData ()** override
 

*Collect data of the Ph Sensor.*
- bool **checkAllgood ()** const override
 

*Check if the Ph Sensor is working properly.*
- std::string **qualifyPh ()** const
 

*This qualifies the Ph Sensor into Acidic, Neutral or Alkaline.*

- void [printData \(\) const override](#)  
*This method prints the data of the Ph Sensor.*
- std::string [stringStatus \(\) const](#)  
*This method returns the status in a string.*
- void [collectAndPrint \(\)](#)  
*Collect and print the data of the Ph Sensor.*

## Friends

- std::ostream & [operator<< \(std::ostream &os, const PhSensor &sensor\)](#)  
*Operator << overload.*

## Additional Inherited Members

### 4.19.1 Detailed Description

Definition at line 15 of file PhSensor.h.

### 4.19.2 Constructor & Destructor Documentation

#### 4.19.2.1 PhSensor()

```
PhSensor::PhSensor (
    int id,
    bool active ) [explicit]
```

Construct a new Ph Sensor object.

##### Parameters

<i>id</i>	
<i>active</i>	

##### Returns

[PhSensor object](#)

Definition at line 9 of file PhSensor.cpp.

```
10      : Sensor(id, Sensor::Types::PH_SENSOR, active) {}
```

### 4.19.2.2 ~PhSensor()

```
PhSensor::~PhSensor ( ) [override]
```

Destroy the Ph Sensor object.

Definition at line 11 of file PhSensor.cpp.

```
11 { }
```

## 4.19.3 Member Function Documentation

### 4.19.3.1 checkAllgood()

```
bool PhSensor::checkAllgood ( ) const [override], [virtual]
```

Check if the Ph Sensor is working properly.

#### Returns

true if the Ph Sensor is working properly  
 false if the Ph Sensor is not working properly

Reimplemented from [Sensor](#).

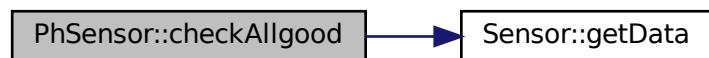
Definition at line 22 of file PhSensor.cpp.

```
22 {
23     float data = Sensor::getData();
24
25     if (data >= 6.2 && data <= 7.8) {
26         return true;
27     } else {
28         return false;
29     }
30 }
```

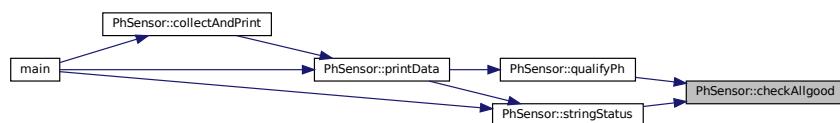
References [Sensor::getData\(\)](#).

Referenced by [qualifyPh\(\)](#), and [stringStatus\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.19.3.2 collectAndPrint()

```
void PhSensor::collectAndPrint ( ) [virtual]
```

Collect and print the data of the Ph Sensor.

Reimplemented from [Sensor](#).

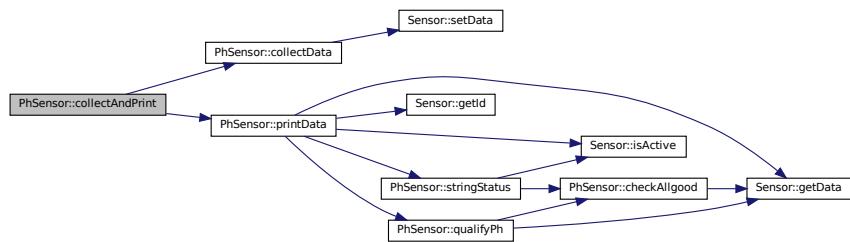
Definition at line 72 of file [PhSensor.cpp](#).

```
72
73     collectData();
74     printData();
75 }
```

References [collectData\(\)](#), and [printData\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.19.3.3 collectData()

```
void PhSensor::collectData ( ) [override], [virtual]
```

Collect data of the Ph Sensor.

This method collects the data of the Ph Sensor and stores it in the data attribute.

Reimplemented from [Sensor](#).

Definition at line 13 of file PhSensor.cpp.

```
13     std::random_device rd;
14     std::mt19937 gen(rd());
15     std::uniform_real_distribution<float> dis(6, 8);
16     float reading = dis(gen);
17
18     Sensor::setData(reading);
19 }
20 }
```

References [Sensor::setData\(\)](#).

Referenced by [collectAndPrint\(\)](#), and [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.19.3.4 printData()

```
void PhSensor::printData() const [override], [virtual]
```

This method prints the data of the Ph Sensor.

Reimplemented from [Sensor](#).

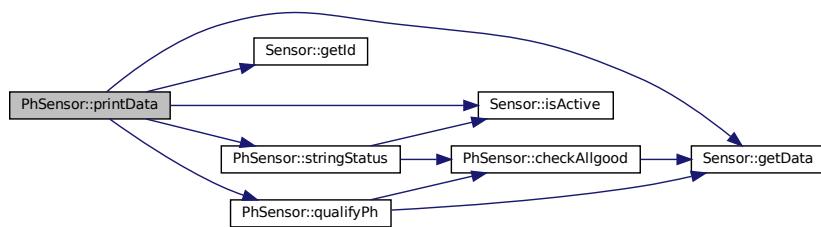
Definition at line 43 of file PhSensor.cpp.

```
43
44     if (Sensor::isActive\(\)) {
45         std::cout << "Ph Sensor with "
46             << "ID: " << Sensor::getId\(\) << " - Data: " << Sensor::getData\(\)
47             << " - Status: " << stringStatus\(\)
48             << " - Qualification: " << qualifyPh\(\) << endl;
49     } else {
50         std::cout << "Ph Sensor ID: " << Sensor::getId\(\)
51             << " - Status: " << stringStatus\(\) << endl;
52     }
53 }
```

References `Sensor::getData()`, `Sensor::getId()`, `Sensor::isActive()`, `qualifyPh()`, and `stringStatus()`.

Referenced by `collectAndPrint()`, and `main()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.19.3.5 `qualifyPh()`

```
std::string PhSensor::qualifyPh ( ) const
```

This qualifies the Ph `Sensor` into Acidic, Neutral or Alkaline.

##### Returns

`std::string` of the qualification of the Ph `Sensor`

Definition at line 32 of file `PhSensor.cpp`.

```

32
33     float data = Sensor::getData();
34     if (this->checkAllgood()) {
35         return "Ideal";
36     } else if (data < 6.5f) {
37         return "Acidic";
38     } else {
39         return "Alkaline";
40     }
41 }
```

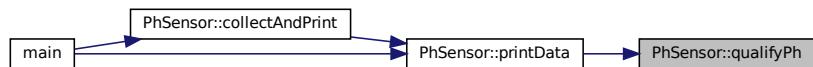
References `checkAllgood()`, and `Sensor::getData()`.

Referenced by `printData()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.19.3.6 stringStatus()

```
std::string PhSensor::stringStatus () const
```

This method returns the status in a string.

##### Returns

`std::string` of the status of the Ph Sensor

Definition at line 60 of file PhSensor.cpp.

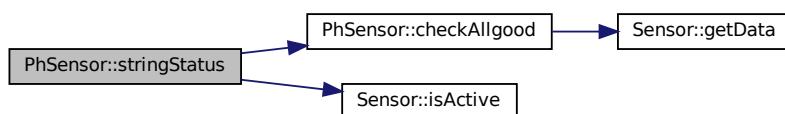
```

60
61     if (Sensor::isActive()) {
62         if (this->checkAllgood()) {
63             return "ACTIVE - GOOD STATUS";
64         } else {
65             return "ACTIVE - BAD STATUS";
66         }
67     } else {
68         return "INACTIVE";
69     }
70 }
```

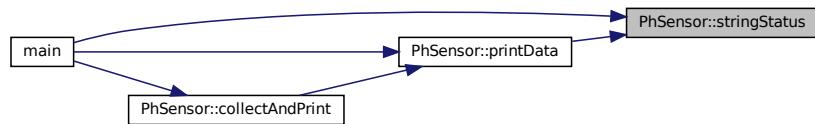
References `checkAllgood()`, and `Sensor::isActive()`.

Referenced by `main()`, and `printData()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.19.4 Friends And Related Function Documentation

##### 4.19.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const PhSensor & sensor ) [friend]
```

Operator << overload.

Definition at line 55 of file PhSensor.cpp.

```
55
56     sensor.printData();
57     return os;
58 }
```

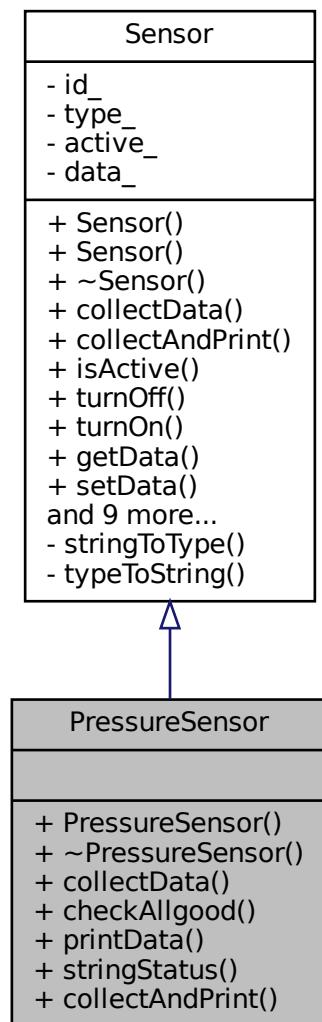
The documentation for this class was generated from the following files:

- src/[PhSensor.h](#)
- src/[PhSensor.cpp](#)

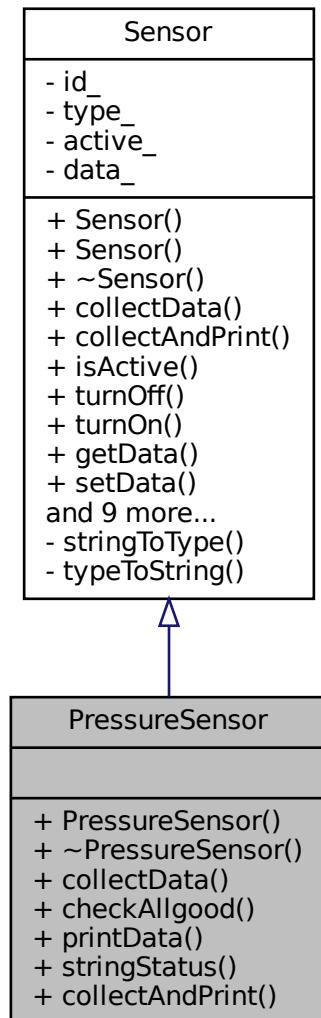
## 4.20 PressureSensor Class Reference

```
#include <PressureSensor.h>
```

Inheritance diagram for PressureSensor:



Collaboration diagram for PressureSensor:



## Public Member Functions

- `PressureSensor (int id, bool active)`  
*Construct a new Pressure `Sensor` object.*
- `~PressureSensor () override`  
*Destroy the Pressure `Sensor` object.*
- `void collectData () override`  
*Collect data of the Pressure `Sensor`.*
- `bool checkAllgood () const override`  
*Check if the Pressure `Sensor` is working properly.*
- `void printData () const override`  
*Print the data of the Pressure `Sensor`.*
- `std::string stringStatus () const`

*This method returns the status in a string.*

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

*Collect and print the data of the Pressure Sensor.*

## Friends

- std::ostream & [operator<< \(std::ostream &os, const PressureSensor &sensor\)](#)

*Operator << overload.*

## Additional Inherited Members

### 4.20.1 Detailed Description

Definition at line 15 of file PressureSensor.h.

### 4.20.2 Constructor & Destructor Documentation

#### 4.20.2.1 PressureSensor()

```
PressureSensor::PressureSensor (
    int id,
    bool active ) [explicit]
```

Construct a new Pressure Sensor object.

##### Parameters

<i>id</i>	
<i>active</i>	

##### Returns

[PressureSensor](#) object

Definition at line 9 of file PressureSensor.cpp.

```
10     : Sensor(id, Sensor::Types::PRESSURE, active) {}
```

#### 4.20.2.2 ~PressureSensor()

```
PressureSensor::~PressureSensor ( ) [override]
```

Destroy the Pressure Sensor object.

Definition at line 12 of file PressureSensor.cpp.

```
12 { }
```

### 4.20.3 Member Function Documentation

#### 4.20.3.1 checkAllgood()

```
bool PressureSensor::checkAllgood ( ) const [override], [virtual]
```

Check if the Pressure Sensor is working properly.

##### Returns

true if the Pressure Sensor is working properly  
 false if the Pressure Sensor is not working properly

Reimplemented from [Sensor](#).

Definition at line 23 of file PressureSensor.cpp.

```
23
24     float data = Sensor::getData();
25     if (data >= 0.91f && data <= 1.09f) {
26         return true;
27     } else {
28         return false;
29     }
30 }
```

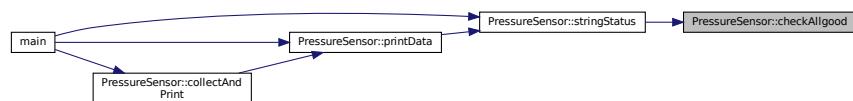
References [Sensor::getData\(\)](#).

Referenced by [stringStatus\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 4.20.3.2 collectAndPrint()

```
void PressureSensor::collectAndPrint ( ) [virtual]
```

Collect and print the data of the Pressure Sensor.

Reimplemented from [Sensor](#).

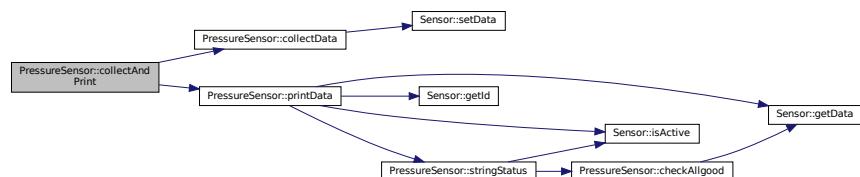
Definition at line 60 of file PressureSensor.cpp.

```
60
61   collectData();
62   printData();
63 }
```

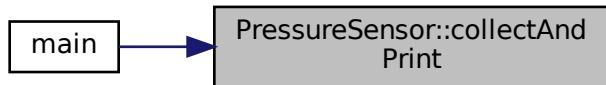
References [collectData\(\)](#), and [printData\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 4.20.3.3 collectData()

```
void PressureSensor::collectData ( ) [override], [virtual]
```

Collect data of the Pressure Sensor.

This method collects the data of the Pressure Sensor and stores it in the data attribute.

Reimplemented from [Sensor](#).

Definition at line 14 of file PressureSensor.cpp.

```

14
15 // Numero random entre 0.90 y 1.10 bares
16 std::random_device rd;
17 std::mt19937 gen(rd());
18 std::uniform_real_distribution<double> dis(0.9, 1.1);
19 float pressure = dis(gen);
20 Sensor::setData(pressure);
21 }

```

References [Sensor::setData\(\)](#).

Referenced by [collectAndPrint\(\)](#), and [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.20.3.4 [printData\(\)](#)

```
void PressureSensor::printData() const [override], [virtual]
```

Print the data of the Pressure [Sensor](#).

Reimplemented from [Sensor](#).

Definition at line 37 of file [PressureSensor.cpp](#).

```

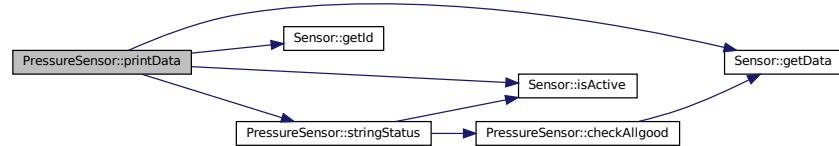
37
38     if (Sensor::isActive\(\)) {
39         std::cout << "Pressure Sensor with "
40             << "ID: " << Sensor::getId\(\) << " - Data: " << Sensor::getData\(\)
41             << " bar - Status: " << stringStatus\(\) << endl;
42     } else {
43         std::cout << "Pressure Sensor ID: " << Sensor::getId\(\)
44             << " - Status: " << stringStatus\(\) << endl;
45     }
46 }

```

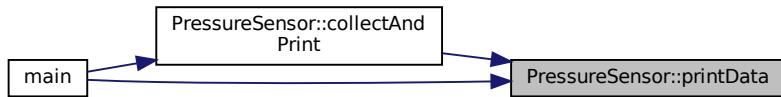
References [Sensor::getData\(\)](#), [Sensor::getId\(\)](#), [Sensor::isActive\(\)](#), and [stringStatus\(\)](#).

Referenced by collectAndPrint(), and main().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.20.3.5 stringStatus()

```
std::string PressureSensor::stringStatus ( ) const
```

This method returns the status in a string.

##### Returns

`std::string` of the status of the Pressure Sensor

Definition at line 48 of file PressureSensor.cpp.

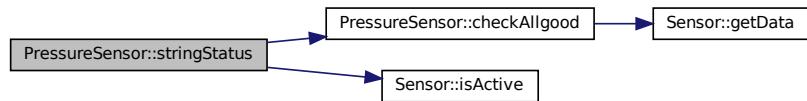
```

48
49     if (Sensor::isActive()) {
50         if (this->checkAllgood()) {
51             return "ACTIVE - GOOD STATUS";
52         } else {
53             return "ACTIVE - BAD STATUS";
54         }
55     } else {
56         return "INACTIVE";
57     }
58 }
```

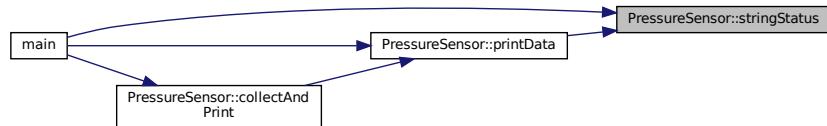
References checkAllgood(), and Sensor::isActive().

Referenced by main(), and printData().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.20.4 Friends And Related Function Documentation

##### 4.20.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const PressureSensor & sensor ) [friend]
```

Operator << overload.

Definition at line 32 of file PressureSensor.cpp.

```
32
33     sensor.printData();
34     return os;
35 }
```

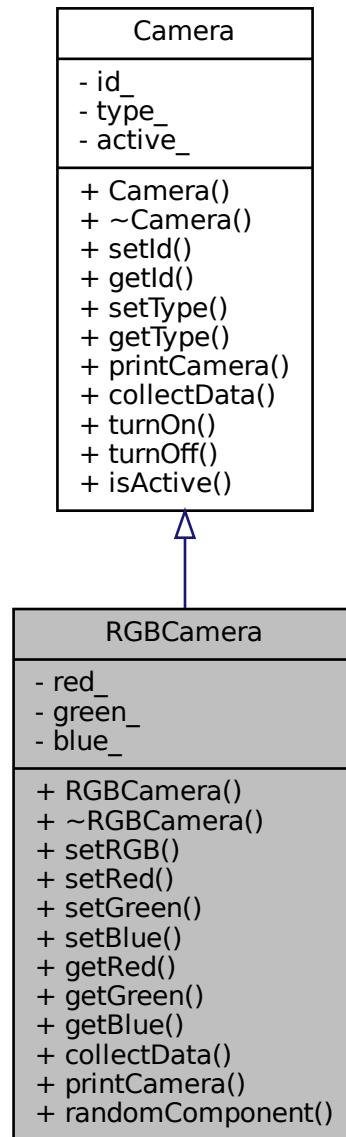
The documentation for this class was generated from the following files:

- src/[PressureSensor.h](#)
- src/[PressureSensor.cpp](#)

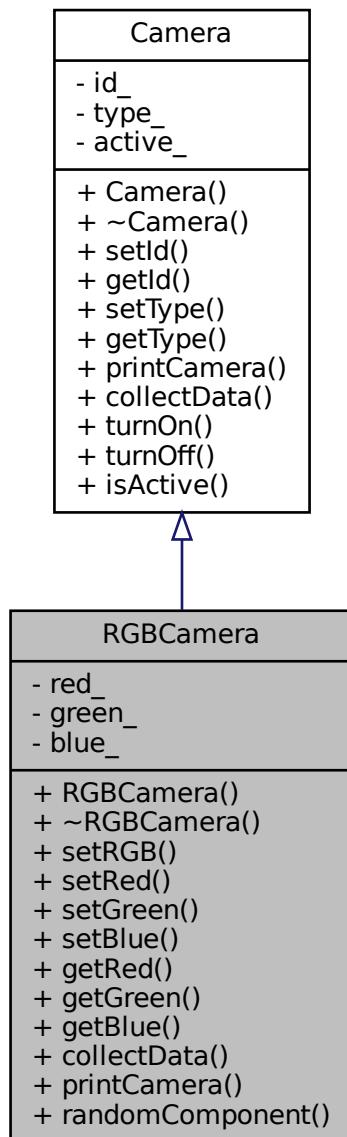
## 4.21 RGBCamera Class Reference

```
#include <RGBCamera.h>
```

Inheritance diagram for RGBCamera:



Collaboration diagram for RGBCamera:



## Public Member Functions

- **RGBCamera** (int id)  
*Construct a new RGB Camera object.*
- **~RGBCamera ()**  
*Destroy the RGB Camera object.*
- void **setRGB** (int red, int green, int blue)  
*This method sets the RGB values of the camera.*
- void **setRed** (int red)  
*This method sets the red value of the camera.*

- void `setGreen` (int green)  
*This method sets the green value of the camera.*
- void `setBlue` (int blue)  
*This method sets the blue value of the camera.*
- int `getRed` ()  
*This method returns the red value of the camera.*
- int `getGreen` ()  
*This method returns the green value of the camera.*
- int `getBlue` ()  
*This method returns the blue value of the camera.*
- void `collectData` () override  
*This method collects the data of the camera.*
- void `printCamera` () override  
*This method prints the camera information.*
- int `randomComponent` ()  
*This method returns a random number between 0 and 255.*

## Private Attributes

- int `red_`  
*This attribute represents the red value of the camera.*
- int `green_`  
*This attribute represents the green value of the camera.*
- int `blue_`  
*This attribute represents the blue value of the camera.*

### 4.21.1 Detailed Description

Definition at line 19 of file RGBCamera.h.

### 4.21.2 Constructor & Destructor Documentation

#### 4.21.2.1 RGBCamera()

```
RGBCamera::RGBCamera (
    int id ) [explicit]
```

Construct a new RGB Camera object.

##### Parameters

<code>id</code>	<input type="text"/>
-----------------	----------------------

**Returns**

[RGBCamera](#) object

Definition at line 6 of file RGBCamera.cpp.

```
6           : Camera(id, "RGB", true) {
7   red_ = -1;
8   green_ = -1;
9   blue_ = -1;
10  std::cout << "RGB Camera id (" << getId() << ")" created" << std::endl;
11 }
```

References `blue_`, `Camera::getId()`, `green_`, and `red_`.

Here is the call graph for this function:



#### 4.21.2.2 ~RGBCamera()

`RGBCamera::~RGBCamera( )`

Destroy the RGB [Camera](#) object.

Definition at line 13 of file RGBCamera.cpp.

```
13 { }
```

### 4.21.3 Member Function Documentation

#### 4.21.3.1 collectData()

`void RGBCamera::collectData( ) [override], [virtual]`

This method collects the data of the camera.

Reimplemented from [Camera](#).

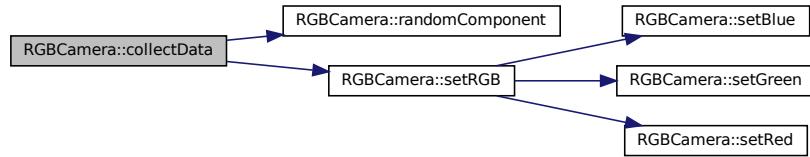
Definition at line 41 of file RGBCamera.cpp.

```
41           {
42   setRGB(randomComponent(), randomComponent(), randomComponent());
43 }
```

References `randomComponent()`, and `setRGB()`.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.21.3.2 getBlue()

```
int RGBCamera::getBlue ( )
```

This method returns the blue value of the camera.

**Returns**

int

Definition at line 31 of file `RGBCamera.cpp`.

```
31 { return blue_; }
```

References `blue_`.

Referenced by `ManageCameras::saveCameras()`.

Here is the caller graph for this function:



#### 4.21.3.3 getGreen()

```
int RGBCamera::getGreen ( )
```

This method returns the green value of the camera.

##### Returns

int

Definition at line 29 of file RGBCamera.cpp.

```
29 { return green_; }
```

References green\_.

Referenced by ManageCameras::saveCameras().

Here is the caller graph for this function:



#### 4.21.3.4 getRed()

```
int RGBCamera::getRed ( )
```

This method returns the red value of the camera.

##### Returns

int

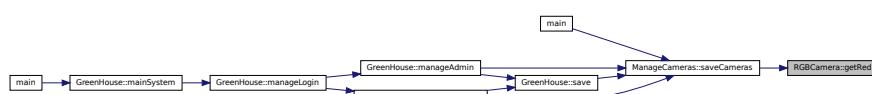
Definition at line 27 of file RGBCamera.cpp.

```
27 { return red_; }
```

References red\_.

Referenced by ManageCameras::saveCameras().

Here is the caller graph for this function:



#### 4.21.3.5 printCamera()

```
void RGBCamera::printCamera ( ) [override], [virtual]
```

This method prints the camera information.

Reimplemented from [Camera](#).

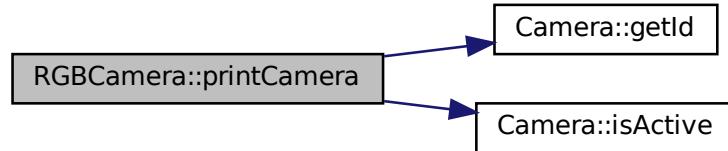
Definition at line 45 of file RGBCamera.cpp.

```
45     {
46     if (isActive() and red_ != -1 and green_ != -1 and blue_ != -1)
47         std::cout << "RGB Camera id (" << getId() << ")"
48         << ":" <<
49         << " red: " << red_ << " green: " << green_ << " blue: " << blue_
50         << std::endl;
51     else if (isActive() and red_ == -1 and green_ == -1 and blue_ == -1)
52         std::cout << "RGB Camera " << getId()
53         << " is active but has not collected data yet" << std::endl;
54     else
55         std::cout << "RGB Camera " << getId() << " is not active" << std::endl;
56 }
```

References blue\_, Camera::getId(), green\_, Camera::isActive(), and red\_.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.21.3.6 randomComponent()

```
int RGBCamera::randomComponent ( )
```

This method returns a random number between 0 and 255.

##### Returns

```
int
```

Definition at line 33 of file RGBCamera.cpp.

```
33 {  
34     // Random number entre 0 y 255  
35     std::random_device rd;  
36     std::mt19937 gen(rd());  
37     std::uniform_int_distribution<> dis(0, 255);  
38     return dis(gen);  
39 }
```

Referenced by `collectData()`.

Here is the caller graph for this function:



#### 4.21.3.7 setBlue()

```
void RGBCamera::setBlue (   
    int blue )
```

This method sets the blue value of the camera.

##### Parameters

<i>blue</i>	
-------------	--

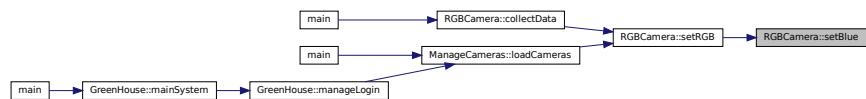
Definition at line 25 of file RGBCamera.cpp.

```
25 { blue_ = blue; }
```

References `blue_`.

Referenced by `setRGB()`.

Here is the caller graph for this function:



#### 4.21.3.8 setGreen()

```
void RGBCamera::setGreen (
    int green )
```

This method sets the green value of the camera.

##### Parameters

<i>green</i>	<input type="text"/>
--------------	----------------------

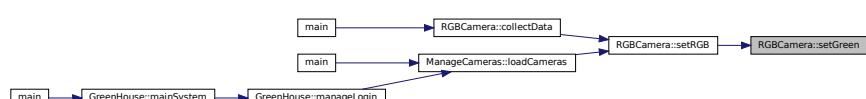
Definition at line 23 of file RGBCamera.cpp.

```
23 { green_ = green; }
```

References *green\_*.

Referenced by *setRGB()*.

Here is the caller graph for this function:



#### 4.21.3.9 setRed()

```
void RGBCamera::setRed (
    int red )
```

This method sets the red value of the camera.

##### Parameters

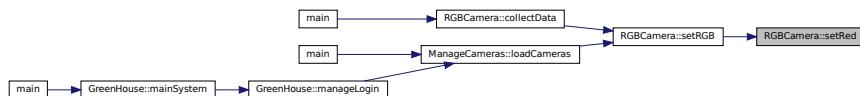
<i>red</i>	<input type="text"/>
------------	----------------------

Definition at line 21 of file RGBCamera.cpp.  
 21 { **red\_** = red; }

References **red\_**.

Referenced by setRGB().

Here is the caller graph for this function:



#### 4.21.3.10 setRGB()

```
void RGBCamera::setRGB (
    int red,
    int green,
    int blue )
```

This method sets the RGB values of the camera.

##### Parameters

<i>red</i>	
<i>green</i>	
<i>blue</i>	

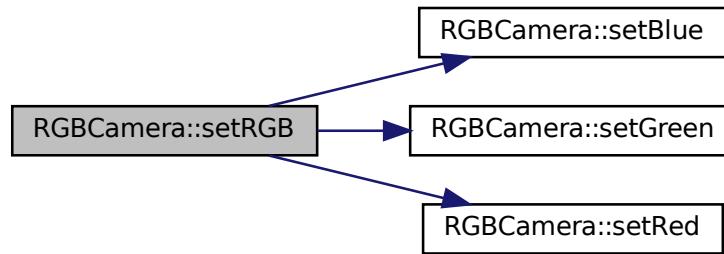
Definition at line 15 of file RGBCamera.cpp.

```
15
16     setRed(red);
17     setGreen(green);
18     setBlue(blue);
19 }
```

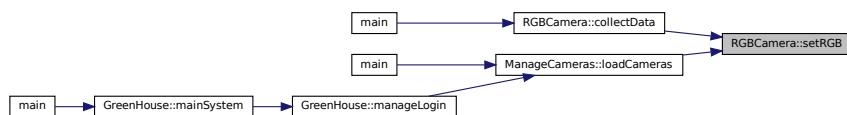
References setBlue(), setGreen(), and setRed().

Referenced by collectData(), and ManageCameras::loadCameras().

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.21.4 Member Data Documentation

### 4.21.4.1 `blue_`

```
int RGBCamera::blue_ [private]
```

This attribute represents the blue value of the camera.

Definition at line 111 of file `RGBCamera.h`.

Referenced by `getBlue()`, `printCamera()`, `RGBCamera()`, and `setBlue()`.

### 4.21.4.2 `green_`

```
int RGBCamera::green_ [private]
```

This attribute represents the green value of the camera.

Definition at line 106 of file `RGBCamera.h`.

Referenced by `getGreen()`, `printCamera()`, `RGBCamera()`, and `setGreen()`.

#### 4.21.4.3 red\_

```
int RGBCamera::red_ [private]
```

This attribute represents the red value of the camera.

Definition at line 101 of file RGBCamera.h.

Referenced by getRed(), printCamera(), RGBCamera(), and setRed().

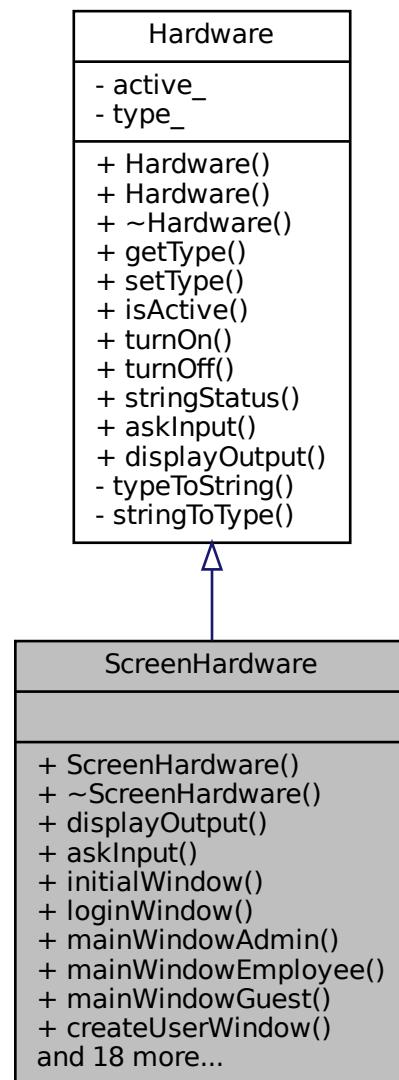
The documentation for this class was generated from the following files:

- src/[RGBCamera.h](#)
- src/[RGBCamera.cpp](#)

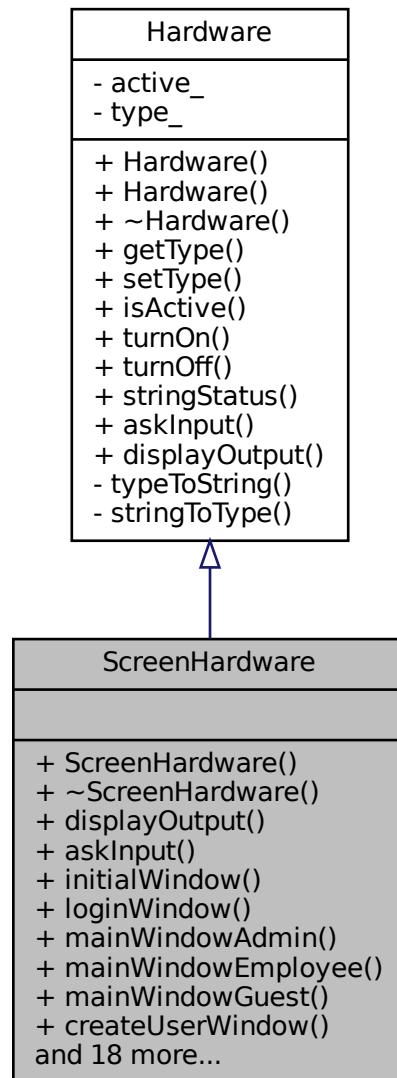
## 4.22 ScreenHardware Class Reference

```
#include <ScreenHardware.h>
```

Inheritance diagram for ScreenHardware:



Collaboration diagram for ScreenHardware:



## Public Member Functions

- [ScreenHardware \(bool active\)](#)  
*Construct a new Screen [Hardware](#) object.*
- [~ScreenHardware \(\) override](#)  
*Destroy the Screen [Hardware](#) object.*
- void [displayOutput \(\) const override](#)  
*This method displays the output of the system.*
- int [askInput \(\) override](#)  
*Ask for an input to the user (this method in reality is not used, we use the keyboard to ask for an input)*
- void [initialWindow \(\)](#)

- void `loginWindow ()`  
*This method displays the initial window of the system.*
- void `mainWindowAdmin ()`  
*This method displays the login window of the system.*
- void `mainWindowEmployee ()`  
*This method displays the main window of the system for the admin.*
- void `mainWindowGuest ()`  
*This method displays the main window of the system for the employee.*
- void `createUserWindow ()`  
*Create a `User` object window.*
- void `deleteUserWindow ()`  
*Delete a `User` object window.*
- void `updateUserWindow ()`  
*Update a `User` object window.*
- void `displayUsersWindow ()`  
*Display all `Users` window.*
- void `saveUsersWindow ()`  
*Save the `users` window.*
- void `createAlarmWindow ()`  
*Create a `Alarm Window` object.*
- void `deleteAlarmWindow ()`  
*Delete a `Alarm Window` object.*
- void `displaySensorsWindow ()`  
*Display all `Sensors` window.*
- void `displayAlarmsWindow ()`  
*Display all `Alarms` window.*
- void `saveAlarmWindow ()`  
*Save `Alarm Window` object.*
- void `turnOnOffSystemWindow ()`  
*Turn on or off the system window.*
- void `displayErrorWindow ()`  
*Display the error window.*
- void `cleanScreen ()`  
*Clean the screen.*
- void `exitWindow ()`  
*Exit the window.*
- void `displayCameraWindow ()`  
*Display the camera window.*
- void `createCameraWindow ()`  
*Create a camera window.*
- void `deleteCameraWindow ()`  
*Delete a camera window.*
- void `turnOnOffCameraWindow ()`  
*Turn on or off the camera window.*
- void `saveCameraWindow ()`  
*Save a camera window.*

## Additional Inherited Members

### 4.22.1 Detailed Description

Definition at line 15 of file ScreenHardware.h.

### 4.22.2 Constructor & Destructor Documentation

#### 4.22.2.1 ScreenHardware()

```
ScreenHardware::ScreenHardware (
    bool active ) [explicit]
```

Construct a new Screen [Hardware](#) object.

##### Parameters

active	<input type="checkbox"/>
--------	--------------------------

##### Returns

[ScreenHardware](#) object

Definition at line 15 of file ScreenHardware.cpp.

```
16     : Hardware(active, Hardware::Types\_Hardware::SCREEN) {}
```

#### 4.22.2.2 ~ScreenHardware()

```
ScreenHardware::~ScreenHardware () [override]
```

Destroy the Screen [Hardware](#) object.

Definition at line 18 of file ScreenHardware.cpp.

```
18 { }
```

### 4.22.3 Member Function Documentation

#### 4.22.3.1 askInput()

```
int ScreenHardware::askInput ( ) [override], [virtual]
```

Ask for an input to the user (this method in reality is not used, we use the keyboard to ask for an input)

##### Returns

```
int
```

Reimplemented from [Hardware](#).

Definition at line 24 of file ScreenHardware.cpp.

```
24
25     std::cout << "Screen wating a input..." << std::endl;
26     return 0;
27 }
```

#### 4.22.3.2 cleanScreen()

```
void ScreenHardware::cleanScreen ( )
```

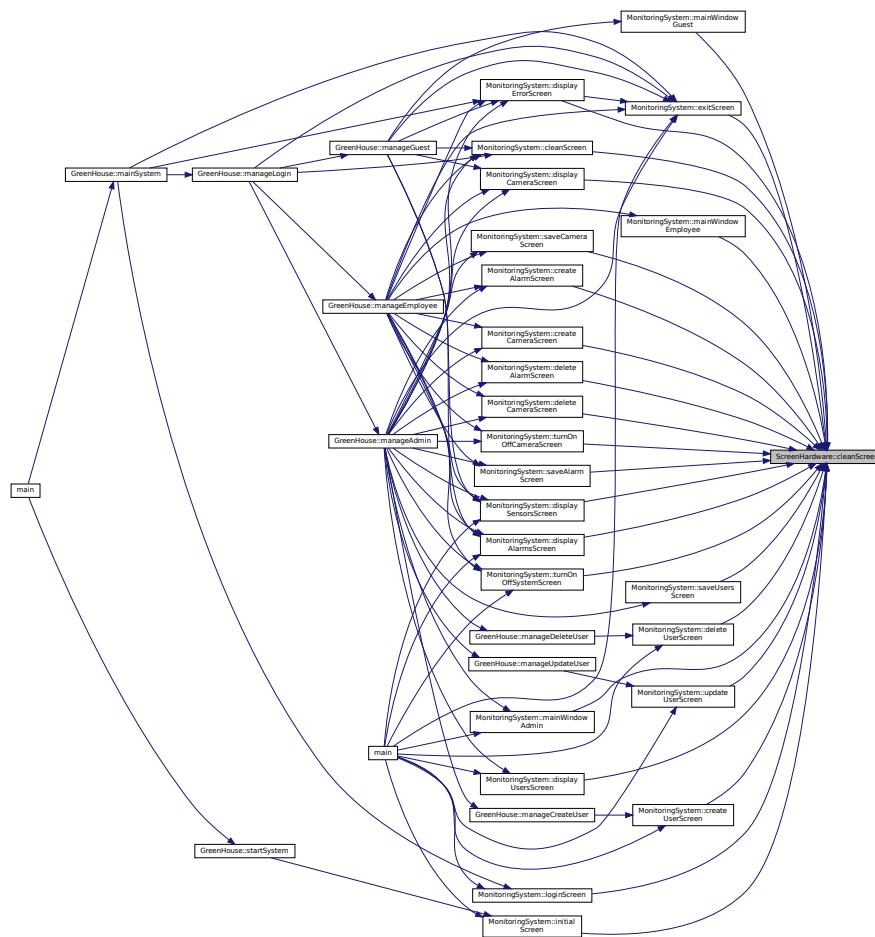
Clean the screen.

Definition at line 206 of file ScreenHardware.cpp.

```
206
207 // Aqui tengo que limpiar la pantalla
208 system("clear");
209 }
```

Referenced by MonitoringSystem::cleanScreen(), MonitoringSystem::createAlarmScreen(), MonitoringSystem::createCameraScreen(), MonitoringSystem::createUserScreen(), MonitoringSystem::deleteAlarmScreen(), MonitoringSystem::deleteCameraScreen(), MonitoringSystem::deleteUserScreen(), MonitoringSystem::displayAlarmsScreen(), MonitoringSystem::displayCameraScreen(), MonitoringSystem::displayErrorScreen(), MonitoringSystem::displaySensorsScreen(), MonitoringSystem::displayUsersScreen(), MonitoringSystem::exitScreen(), MonitoringSystem::initialScreen(), MonitoringSystem::loginScreen(), MonitoringSystem::mainWindowAdmin(), MonitoringSystem::mainWindowEmployee(), MonitoringSystem::mainWindowGuest(), MonitoringSystem::saveAlarmScreen(), MonitoringSystem::saveCameraScreen(), MonitoringSystem::saveUsersScreen(), MonitoringSystem::turnOnOffCameraScreen(), MonitoringSystem::turnOnOffSystemScreen(), and MonitoringSystem::updateUserScreen().

Here is the caller graph for this function:



#### 4.22.3.3 createAlarmWindow()

```
void ScreenHardware::createAlarmWindow ( )
```

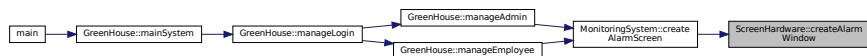
Create a Alarm Window object.

Definition at line 173 of file ScreenHardware.cpp.

```
173   {
174     // Aqui tengo que mostrar un menu para crear una alarma
175     std::cout << "----_Create Alarm Window----" << std::endl;
176     std::cout << "First the type(intro), then the id(intro)" << std::endl;
177 }
```

Referenced by MonitoringSystem::createAlarmScreen().

Here is the caller graph for this function:



#### 4.22.3.4 createCameraWindow()

```
void ScreenHardware::createCameraWindow ( )
```

Create a camera window.

Definition at line 222 of file ScreenHardware.cpp.

```
222  {
223  // Aqui tengo que mostrar un menu para crear una camara
224  std::cout << "___Create Camera Window___" << std::endl;
225  std::cout << "Creating a new camera..." << std::endl;
226 }
```

Referenced by MonitoringSystem::createCameraScreen().

Here is the caller graph for this function:



#### 4.22.3.5 createUserWindow()

```
void ScreenHardware::createUserWindow ( )
```

Create a [User](#) object window.

Definition at line 143 of file ScreenHardware.cpp.

```
143  {
144  // Aqui tengo que mostrar un menu para crear un usuario
145  std::cout << "___Create User Window___" << std::endl;
146  std::cout << ASK_DATA << std::endl;
147  std::cout << USER_PROMPT << std::endl;
148 }
```

References ASK\_DATA, and USER\_PROMPT.

Referenced by MonitoringSystem::createUserScreen(), and main().

Here is the caller graph for this function:



#### 4.22.3.6 deleteAlarmWindow()

```
void ScreenHardware::deleteAlarmWindow ( )
```

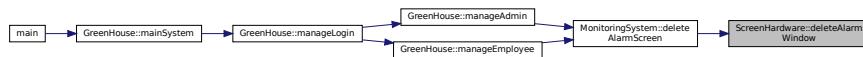
Delete a Alarm Window object.

Definition at line 178 of file ScreenHardware.cpp.

```
178  {
179  // Aqui tengo que mostrar un menu para borrar una alarma
180  std::cout << "----Delete Alarm Window----" << std::endl;
181  std::cout << "Enter the id of the alarm you want to delete" << std::endl;
182 }
```

Referenced by MonitoringSystem::deleteAlarmScreen().

Here is the caller graph for this function:



#### 4.22.3.7 deleteCameraWindow()

```
void ScreenHardware::deleteCameraWindow ( )
```

Delete a camera window.

Definition at line 228 of file ScreenHardware.cpp.

```
228  {
229  // Aqui tengo que mostrar un menu para borrar una camara
230  std::cout << "----Delete Camera Window----" << std::endl;
231  std::cout << "Deleting a camera..." << std::endl;
232 }
```

Referenced by MonitoringSystem::deleteCameraScreen().

Here is the caller graph for this function:



#### 4.22.3.8 deleteUserWindow()

```
void ScreenHardware::deleteUserWindow ( )
```

Delete a [User](#) object window.

Definition at line 150 of file ScreenHardware.cpp.

```
150
151 // Aqui tengo que mostrar un menu para borrar un usuario
152 std::cout << "___Delete User Window___" << std::endl;
153 std::cout << "First the name(intro), then the password(intro), then the "
154           "nif(intro)"
155           << std::endl;
156 }
```

Referenced by [MonitoringSystem::deleteUserScreen\(\)](#), and [main\(\)](#).

Here is the caller graph for this function:



#### 4.22.3.9 displayAlarmsWindow()

```
void ScreenHardware::displayAlarmsWindow ( )
```

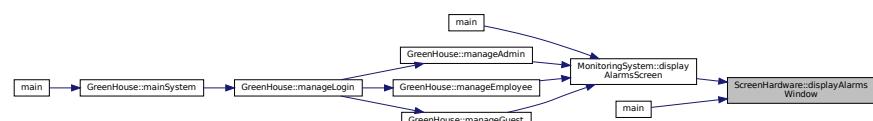
Display all Alarms window.

Definition at line 189 of file ScreenHardware.cpp.

```
189
190 // Aqui tengo que mostrar un menu para ver todas las alarmas
191 std::cout << "___Display Alarms Window___" << std::endl;
192 }
```

Referenced by [MonitoringSystem::displayAlarmsScreen\(\)](#), and [main\(\)](#).

Here is the caller graph for this function:



#### 4.22.3.10 displayCameraWindow()

```
void ScreenHardware::displayCameraWindow ( )
```

Display the camera window.

Definition at line 217 of file ScreenHardware.cpp.

```
217  {
218  // Aqui tengo que mostrar un menu para ver la camara
219  std::cout << "___Display Camera Window___" << std::endl;
220 }
```

Referenced by MonitoringSystem::displayCameraScreen().

Here is the caller graph for this function:



#### 4.22.3.11 displayErrorWindow()

```
void ScreenHardware::displayErrorWindow ( )
```

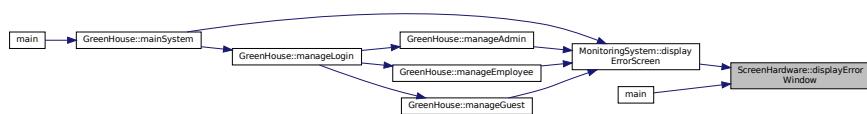
Display the error window.

Definition at line 200 of file ScreenHardware.cpp.

```
200  {
201  // Aqui tengo que mostrar un menu de error
202  std::cout << "___Display Error Window___" << std::endl;
203  std::cout << "And error happend" << std::endl;
204 }
```

Referenced by MonitoringSystem::displayErrorScreen(), and main().

Here is the caller graph for this function:



#### 4.22.3.12 displayOutput()

```
void ScreenHardware::displayOutput ( ) const [override], [virtual]
```

This method displays the output of the system.

Reimplemented from [Hardware](#).

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

```
20     {
21     std::cout << "Displaying output..." << std::endl;
22 }
```

#### 4.22.3.13 displaySensorsWindow()

```
void ScreenHardware::displaySensorsWindow ( )
```

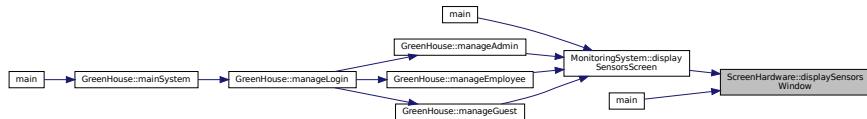
Display all Sensors window.

Definition at line 184 of file [ScreenHardware.cpp](#).

```
184     {
185     // Aquí tengo que mostrar un menu para ver todos los sensores
186     std::cout << "---_Display Sensors Window_---" << std::endl;
187 }
```

Referenced by [MonitoringSystem::displaySensorsScreen\(\)](#), and [main\(\)](#).

Here is the caller graph for this function:



#### 4.22.3.14 displayUsersWindow()

```
void ScreenHardware::displayUsersWindow ( )
```

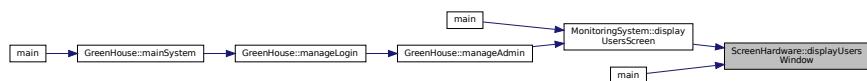
Display all Users window.

Definition at line 168 of file [ScreenHardware.cpp](#).

```
168     {
169     // Aquí tengo que mostrar un menu para ver todos los usuarios
170     std::cout << "---_Display Users Window_---" << std::endl;
171 }
```

Referenced by [MonitoringSystem::displayUsersScreen\(\)](#), and [main\(\)](#).

Here is the caller graph for this function:



#### 4.22.3.15 exitWindow()

```
void ScreenHardware::exitWindow ( )
```

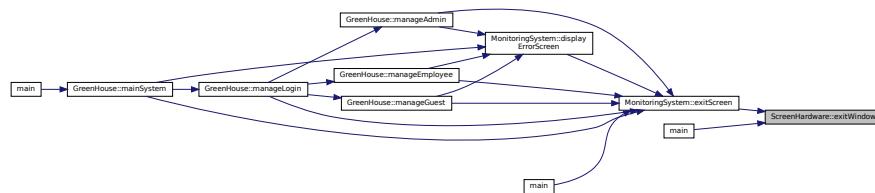
Exit the window.

Definition at line 36 of file ScreenHardware.cpp.

```
36
37 // Aquí es el menu de salida
38 std::cout << "___Exit Window___" << std::endl;
39 std::cout << "Thanks for using our system" << std::endl;
40 }
```

Referenced by MonitoringSystem::exitScreen(), and main().

Here is the caller graph for this function:



#### 4.22.3.16 initialWindow()

```
void ScreenHardware::initialWindow ( )
```

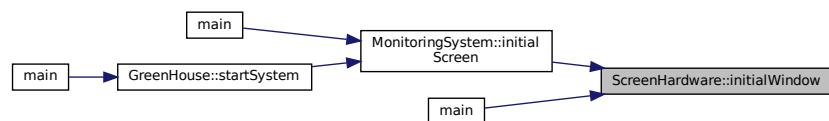
This method displays the initial window of the system.

Definition at line 29 of file ScreenHardware.cpp.

```
29
30 // Aquí tengo que mostrar un menu principal donde se de la bienvenida
31 std::cout << "___Initial Window___" << std::endl;
32 std::cout << "1. Login" << std::endl;
33 std::cout << "2. Exit" << std::endl;
34 }
```

Referenced by MonitoringSystem::initialScreen(), and main().

Here is the caller graph for this function:



#### 4.22.3.17 loginWindow()

```
void ScreenHardware::loginWindow ( )
```

This method displays the login window of the system.

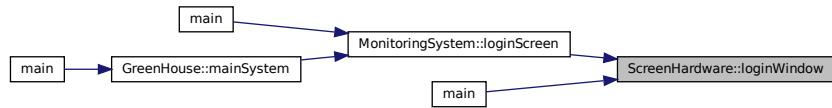
Definition at line 42 of file ScreenHardware.cpp.

```
42
43     // Este es el menu de login
44     std::cout << "___Login Window___" << std::endl;
45     std::cout << ASK_DATA << std::endl;
46     std::cout
47         << "First the name(intro), then the password(intro), then the nif(intro)"
48         << std::endl;
49 }
```

References ASK\_DATA.

Referenced by MonitoringSystem::loginScreen(), and main().

Here is the caller graph for this function:



#### 4.22.3.18 mainWindowAdmin()

```
void ScreenHardware::mainWindowAdmin ( )
```

This method displays the main window of the system for the admin.

Definition at line 51 of file ScreenHardware.cpp.

```
51
52     // Aqui tengo que mostrar un menu principal donde se muestran todas las
53     // opciones para admins
54     std::cout << "___Main Window Admin___" << std::endl;
55     std::cout << "\n" << std::endl;
56     // Seccion de usuarios
57     std::cout << "-Users Section-" << std::endl;
58     std::cout << "-----" << std::endl;
59     std::cout << "1. Create User" << std::endl;
60     std::cout << "2. Delete User" << std::endl;
61     std::cout << "3. Update User" << std::endl;
62     std::cout << "4. Display Users" << std::endl;
63     std::cout << "-----" << std::endl;
64     std::cout << "\n" << std::endl;
65     // Seccion de alarmas y sensores
66     std::cout << "-Alarms/Sensors Section-" << std::endl;
67     std::cout << "-----" << std::endl;
68     std::cout << "5. Create Sensor" << std::endl;
69     std::cout << "6. Delete Sensor" << std::endl;
70     std::cout << "7. Display Sensors" << std::endl;
71     std::cout << "8. Display Alarms" << std::endl;
72     std::cout << "9. Turn On/Off Sensors/Alarms" << std::endl;
73     std::cout << "-----" << std::endl;
74     std::cout << "\n" << std::endl;
75     // Seccion de camaras
76     std::cout << "-Cameras Section-" << std::endl;
77     std::cout << "-----" << std::endl;
78     std::cout << "10. Display Cameras" << std::endl;
```

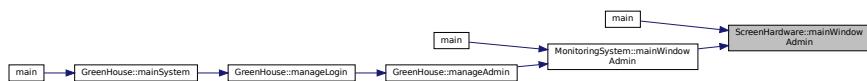
```

79 std::cout << "11. Create Camera" << std::endl;
80 std::cout << "12. Delete Camera" << std::endl;
81 std::cout << "13. Turn On/Off Cameras" << std::endl;
82 std::cout << "-----" << std::endl;
83 std::cout << "\n" << std::endl;
84 // Seccion de guardar y salir
85 std::cout << "-Save/Exit Section-" << std::endl;
86 std::cout << "-----" << std::endl;
87 std::cout << "14. Save Cameras" << std::endl;
88 std::cout << "15. Save Users" << std::endl;
89 std::cout << "16. Save Sensors" << std::endl;
90 std::cout << "17. Exit & Save all" << std::endl;
91 std::cout << "-----" << std::endl;
92 std::cout << "\n" << std::endl;
93 }

```

Referenced by main(), and MonitoringSystem::mainWindowAdmin().

Here is the caller graph for this function:



#### 4.22.3.19 mainWindowEmployee()

```
void ScreenHardware::mainWindowEmployee ( )
```

This method displays the main window of the system for the employee.

Definition at line 95 of file ScreenHardware.cpp.

```

95 {
96 // Aqui tengo que mostrar un menu principal donde se muestren todas las
97 // opciones para employees
98 std::cout << "___Main Window Employee___" << std::endl;
99 std::cout << "\n" << std::endl;
100 // Seccion de sensores y alarmas
101 std::cout << "-Sensors/Alarms Section-" << std::endl;
102 std::cout << "-----" << std::endl;
103 std::cout << "1. Create Sensor" << std::endl;
104 std::cout << "2. Delete Sensor" << std::endl;
105 std::cout << "3. Display Sensors" << std::endl;
106 std::cout << "4. Display Alarms" << std::endl;
107 std::cout << "5. Turn On/Off Sensors/Alarms" << std::endl;
108 std::cout << "-----" << std::endl;
109 std::cout << "\n" << std::endl;
110 // Seccion de camaras
111 std::cout << "-Cameras Section-" << std::endl;
112 std::cout << "-----" << std::endl;
113 std::cout << "6. Display Cameras" << std::endl;
114 std::cout << "7. Create Camera" << std::endl;
115 std::cout << "8. Delete Camera" << std::endl;
116 std::cout << "9. Turn On/Off Cameras" << std::endl;
117 std::cout << "-----" << std::endl;
118 std::cout << "\n" << std::endl;
119 // Seccion de guardar y salir
120 std::cout << "-Save/Exit Section-" << std::endl;
121 std::cout << "-----" << std::endl;
122 std::cout << "10. Save Cameras" << std::endl;
123 std::cout << "11. Save Sensors" << std::endl;
124 std::cout << "12. Exit & Save all" << std::endl;
125 std::cout << "-----" << std::endl;
126 }

```

Referenced by main(), and MonitoringSystem::mainWindowEmployee().

Here is the caller graph for this function:



#### 4.22.3.20 mainWindowGuest()

```
void ScreenHardware::mainWindowGuest ( )
```

This method displays the main window of the system for the guest.

Definition at line 128 of file ScreenHardware.cpp.

```
128
129 // Aqui tengo que mostrar un menu principal donde se muestran todas las
130 // opciones para guests
131 std::cout << "----_Main Window Guest_---" << std::endl;
132 std::cout << "\n" << std::endl;
133 // Seccion de sensores y alarmas
134 std::cout << "-Sensors/Alarms Section-" << std::endl;
135 std::cout << "-----" << std::endl;
136 std::cout << "1. Display Sensors" << std::endl;
137 std::cout << "2. Display Alarms" << std::endl;
138 std::cout << "3. Display Cameras" << std::endl;
139 std::cout << "4. Exit" << std::endl;
140 std::cout << "-----" << std::endl;
141 }
```

Referenced by main(), and MonitoringSystem::mainWindowGuest().

Here is the caller graph for this function:



#### 4.22.3.21 saveAlarmWindow()

```
void ScreenHardware::saveAlarmWindow ( )
```

Save Alarm Window object.

Definition at line 211 of file ScreenHardware.cpp.

```
211
212 // Aqui tengo que mostrar un menu para guardar una alarma
213 std::cout << "----_Save Alarm Window_---" << std::endl;
214 std::cout << "Saving all the sensors(txt/dat)..." << std::endl;
215 }
```

Referenced by MonitoringSystem::saveAlarmScreen().

Here is the caller graph for this function:



#### 4.22.3.22 saveCameraWindow()

```
void ScreenHardware::saveCameraWindow ( )
```

Save a camera window.

Definition at line 240 of file ScreenHardware.cpp.

```
240      {
241      // Aqui tengo que mostrar un menu para guardar una camara
242      std::cout << "___Save Camera Window___" << std::endl;
243      std::cout << "Saving all the cameras(txt)..." << std::endl;
244  }
```

Referenced by MonitoringSystem::saveCameraScreen().

Here is the caller graph for this function:



#### 4.22.3.23 saveUsersWindow()

```
void ScreenHardware::saveUsersWindow ( )
```

Save the users window.

Definition at line 246 of file ScreenHardware.cpp.

```
246      {
247      // Aqui tengo que mostrar un menu para guardar un usuario
248      std::cout << "___Save User Window___" << std::endl;
249      std::cout << "Saving all the users(txt/dat)..." << std::endl;
250  }
```

Referenced by MonitoringSystem::saveUsersScreen().

Here is the caller graph for this function:



#### 4.22.3.24 turnOnOffCameraWindow()

```
void ScreenHardware::turnOnOffCameraWindow ( )
```

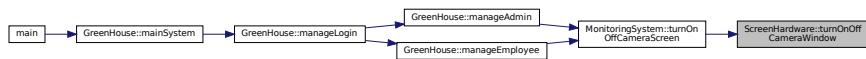
Turn on or off the camera window.

Definition at line 234 of file ScreenHardware.cpp.

```
234 {  
235 // Aqui tengo que mostrar un menu para encender y apagar la camara  
236 std::cout << "----_Turn On/Off Camera Window_---" << std::endl;  
237 std::cout << "1. ON \n2. OFF" << std::endl;  
238 }
```

Referenced by MonitoringSystem::turnOnOffCameraScreen().

Here is the caller graph for this function:



#### 4.22.3.25 turnOnOffSystemWindow()

```
void ScreenHardware::turnOnOffSystemWindow ( )
```

Turn on or off the system window.

Definition at line 194 of file ScreenHardware.cpp.

```
194 {  
195 // Aqui tengo que mostrar un menu para encender y apagar el sistema  
196 std::cout << "----_Turn On/Off System Window_---" << std::endl;  
197 std::cout << "1. ON \n2. OFF" << std::endl;  
198 }
```

Referenced by main(), and MonitoringSystem::turnOnOffSystemScreen().

Here is the caller graph for this function:



#### 4.22.3.26 updateUserWindow()

```
void ScreenHardware::updateUserWindow ( )
```

Update a [User](#) object window.

Definition at line 158 of file ScreenHardware.cpp.

```
158
159 // Aqui tengo que mostrar un menu para actualizar un usuario
160 std::cout << "----_Update User Window_---" << std::endl;
161 std::cout << ASK_DATA << std::endl;
162 std::cout << "You can change the role / the email / if you want to change "
163     "password delete the user and create a new one"
164     << std::endl;
165 std::cout << USER_PROMPT << std::endl;
166 }
```

References ASK\_DATA, and USER\_PROMPT.

Referenced by main(), and MonitoringSystem::updateUserScreen().

Here is the caller graph for this function:



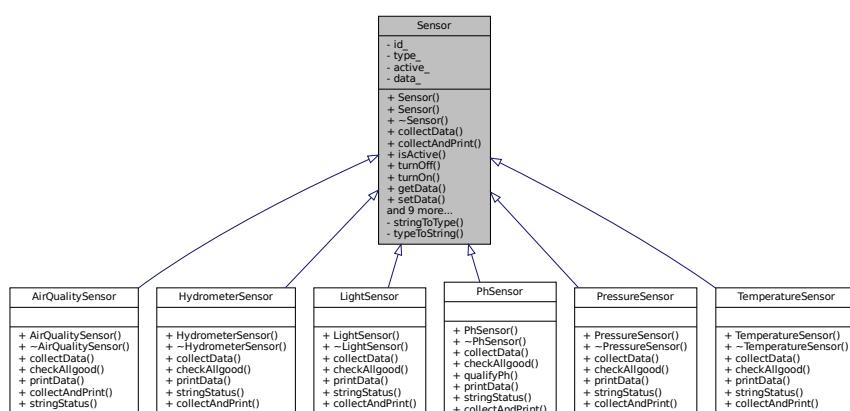
The documentation for this class was generated from the following files:

- src/[ScreenHardware.h](#)
- src/[ScreenHardware.cpp](#)

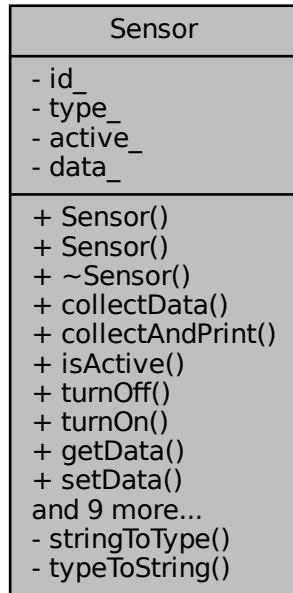
## 4.23 Sensor Class Reference

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

Inheritance diagram for Sensor:



Collaboration diagram for Sensor:



## Public Types

- enum `Types` {  
  `NONE` , `TEMPERATURE` , `AIR_QUALITY` , `HYDROMETER` ,  
  `PRESSURE` , `LIGHT_SENSOR` , `PH_SENSOR` }

*This is the enum `Types`. It contains the types of the sensors.*

## Public Member Functions

- `Sensor ()`  
*Construct a new `Sensor` object.*
- `Sensor (int id, Types type, bool active)`  
*Construct a new `Sensor` object.*
- `virtual ~Sensor ()`  
*Destroy the `Sensor` object.*
- `virtual void collectData ()`  
*Collect data of the `Sensor`.*
- `virtual void collectAndPrint ()`  
*Collect and print the data of the `Sensor`.*
- `bool isActive () const`  
*Return if is active or not the sensor.*
- `void turnOff ()`  
*Turn off the sensor.*
- `void turnOn ()`

- float `getData () const`  
*Get the Data object.*
- void `setData (float data)`  
*Set the Data object.*
- int `getId () const`  
*Get the Id object.*
- void `setId (int newid)`  
*Set the Id object.*
- std::string `getType () const`  
*Get the Type object.*
- void `setType (std::string newtype)`  
*Set the Type object.*
- virtual bool `checkAllgood () const`  
*Check if the Sensor is working properly.*
- bool `operator< (const Sensor &Sensor) const`  
*Operator < overload.*
- bool `operator> (const Sensor &Sensor) const`  
*Operator > overload.*
- bool `operator== (const Sensor &Sensor) const`  
*Operator == overload.*
- virtual void `printData () const`  
*Print the data of the Sensor.*

## Private Member Functions

- Types `stringToType (const std::string &type) const`  
*Convert the string to the type.*
- std::string `typeToString (Types type) const`  
*Convert the type to the string.*

## Private Attributes

- int `id_`  
*The id of the sensor.*
- Types `type_`  
*The type of the sensor.*
- bool `active_`  
*The state of the sensor.*
- float `data_`  
*The data of the sensor.*

## Friends

- std::ostream & `operator<< (std::ostream &os, const Sensor &Sensor)`  
*Operator << overload.*
- std::istream & `operator>> (std::istream &is, Sensor &Sensor)`  
*Operator >> overload.*

### 4.23.1 Detailed Description

Definition at line 13 of file Sensor.h.

### 4.23.2 Member Enumeration Documentation

#### 4.23.2.1 Types

```
enum Sensor::Types
```

This is the enum Types. It contains the types of the sensors.

Enumerator

NONE	
TEMPERATURE	
AIR_QUALITY	
HYDROMETER	
PRESSURE	
LIGHT_SENSOR	
PH_SENSOR	

Definition at line 19 of file Sensor.h.

```
19
20     NONE,
21     TEMPERATURE,
22     AIR_QUALITY,
23     HYDROMETER,
24     PRESSURE,
25     LIGHT_SENSOR,
26     PH_SENSOR,
27 }
```

### 4.23.3 Constructor & Destructor Documentation

#### 4.23.3.1 Sensor() [1/2]

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

Construct a new [Sensor](#) object.

Creates a new [Sensor](#) object with the default values (id, type, active).

**Returns**

[Sensor object](#)

Definition at line 6 of file Sensor.cpp.

```
6     {
7     id_ = -1;
8     type_ = Types::NONE;
9     active_ = false;
10    data_ = -1;
11 }
```

#### 4.23.3.2 Sensor() [2/2]

```
Sensor::Sensor (
    int id,
    Types type,
    bool active ) [explicit]
```

Construct a new [Sensor](#) object.

Creates a new [Sensor](#) object with the values passed as parameters.

**Parameters**

<i>id</i>	of the sensor
<i>type</i>	of the sensor
<i>active</i>	of the sensor

**Returns**

[Sensor object](#)

Definition at line 13 of file Sensor.cpp.

```
13     {
14     id_ = id;
15     type_ = type;
16     active_ = active;
17     data_ = -1;
18 }
```

#### 4.23.3.3 ~Sensor()

```
Sensor::~Sensor ( ) [virtual]
```

Destroy the [Sensor](#) object.

Definition at line 20 of file Sensor.cpp.

```
20 { }
```

### 4.23.4 Member Function Documentation

#### 4.23.4.1 checkAllgood()

```
bool Sensor::checkAllgood ( ) const [virtual]
```

Check if the [Sensor](#) is working properly.

##### Returns

true if the [Sensor](#) is working properly  
false if the [Sensor](#) is not working properly

Reimplemented in [TemperatureSensor](#), [PressureSensor](#), [PhSensor](#), [LightSensor](#), [HydrometerSensor](#), and [AirQualitySensor](#).

Definition at line 128 of file Sensor.cpp.

```
128 { return true; }
```

Referenced by main().

Here is the caller graph for this function:



#### 4.23.4.2 collectAndPrint()

```
void Sensor::collectAndPrint ( ) [virtual]
```

Collect and print the data of the [Sensor](#).

Reimplemented in [TemperatureSensor](#), [PressureSensor](#), [PhSensor](#), [LightSensor](#), [HydrometerSensor](#), and [AirQualitySensor](#).

Definition at line 31 of file Sensor.cpp.

```
31 {
32     collectData();
33     printData();
34 }
```

#### 4.23.4.3 collectData()

```
void Sensor::collectData ( ) [virtual]
```

Collect data of the [Sensor](#).

Reimplemented in [TemperatureSensor](#), [PressureSensor](#), [PhSensor](#), [LightSensor](#), [HydrometerSensor](#), and [AirQualitySensor](#).

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

```
22     {
23     cout << "Collecting data from sensor id" << id_ << " which is: " << getType()
24     << endl;
25     setData(-10000);
26     // This function will be implemented in the derived classes
27     // en la clase derivada se implementara la funcion, generaremos de manera
28     // aleatoria el valor y despues lo asignaremos al atributo data_ con setData()
29 }
```

Referenced by [main\(\)](#).

Here is the caller graph for this function:



#### 4.23.4.4 getData()

```
float Sensor::getData ( ) const
```

Get the Data object.

##### Returns

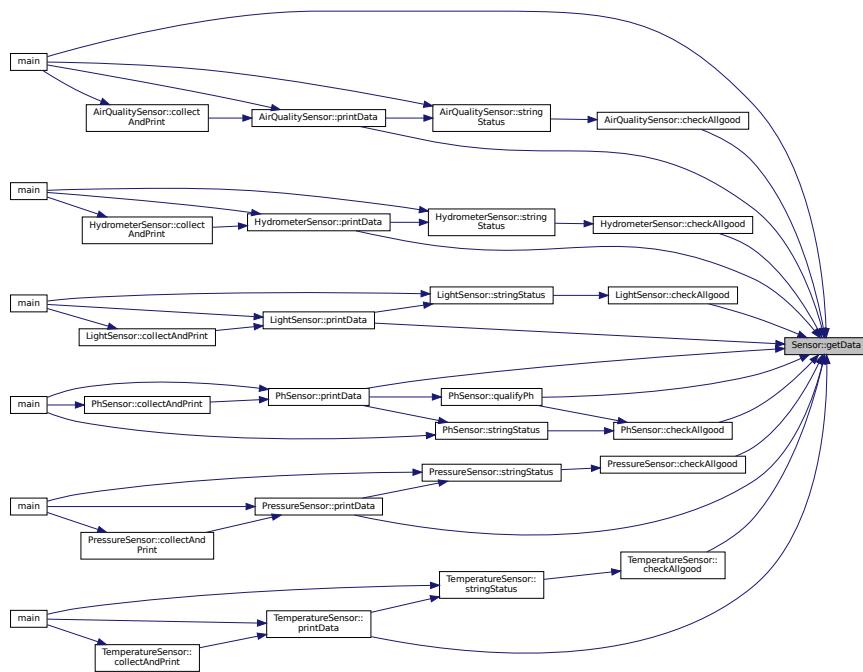
float

Definition at line 42 of file [Sensor.cpp](#).

```
42 { return data_; }
```

Referenced by [AirQualitySensor::checkAllgood\(\)](#), [HydrometerSensor::checkAllgood\(\)](#), [LightSensor::checkAllgood\(\)](#), [PhSensor::checkAllgood\(\)](#), [PressureSensor::checkAllgood\(\)](#), [TemperatureSensor::checkAllgood\(\)](#), [main\(\)](#), [AirQualitySensor::printData\(\)](#), [HydrometerSensor::printData\(\)](#), [LightSensor::printData\(\)](#), [PhSensor::printData\(\)](#), [PressureSensor::printData\(\)](#), [TemperatureSensor::printData\(\)](#), and [PhSensor::qualifyPh\(\)](#).

Here is the caller graph for this function:



#### 4.23.4.5 getId()

```
int Sensor::getId ( ) const
```

Get the Id object.

##### Returns

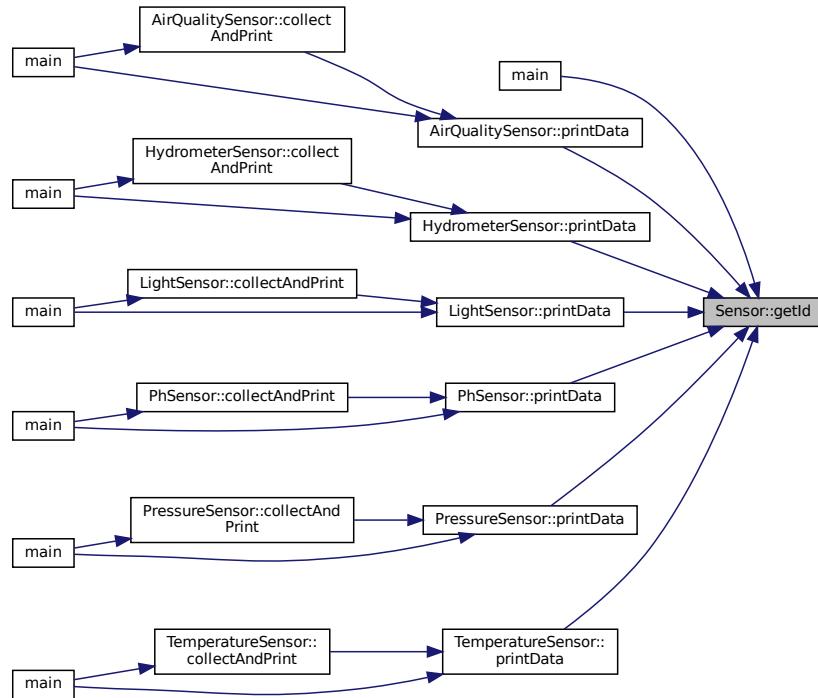
```
int
```

Definition at line 46 of file `Sensor.cpp`.

```
46 { return id_; }
```

Referenced by `main()`, `AirQualitySensor::printData()`, `HydrometerSensor::printData()`, `LightSensor::printData()`, `PhSensor::printData()`, `PressureSensor::printData()`, and `TemperatureSensor::printData()`.

Here is the caller graph for this function:



#### 4.23.4.6 `getType()`

```
std::string Sensor::getType ( ) const
```

Get the Type object.

**Returns**

```
std::string
```

Definition at line 50 of file `Sensor.cpp`.

```
50
51     std::string type = typeToString(type_);
52     return type;
53 }
```

Referenced by `main()`.

Here is the caller graph for this function:



#### 4.23.4.7 isActive()

```
bool Sensor::isActive ( ) const
```

Return if is active or not the sensor.

Returns

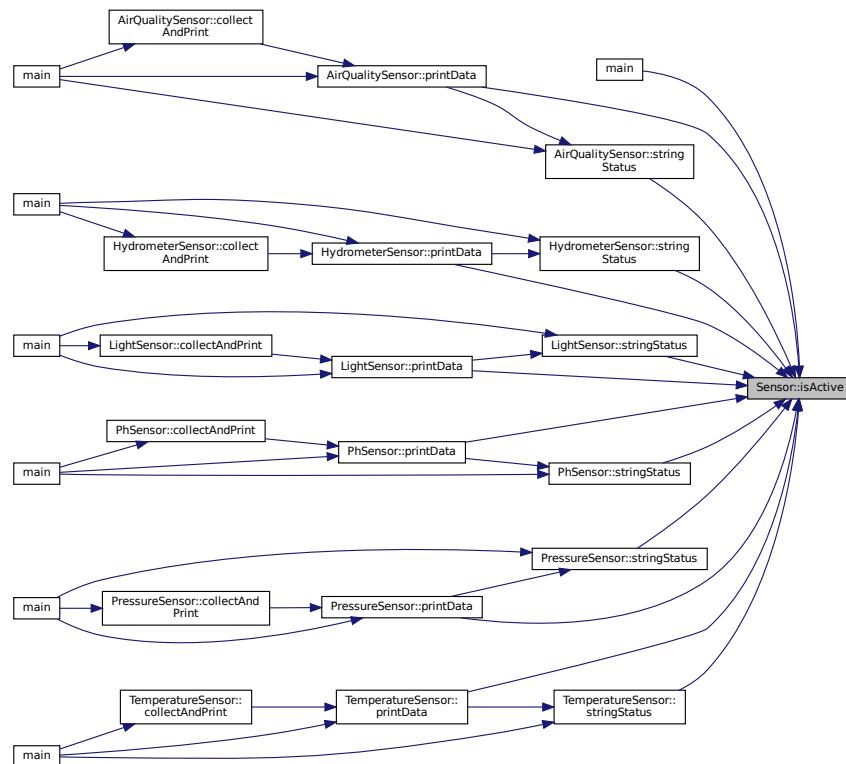
```
true  
false
```

Definition at line 36 of file Sensor.cpp.

```
36 { return active_; }
```

Referenced by main(), AirQualitySensor::printData(), HydrometerSensor::printData(), LightSensor::printData(), PhSensor::printData(), PressureSensor::printData(), TemperatureSensor::printData(), AirQualitySensor::stringStatus(), HydrometerSensor::stringStatus(), LightSensor::stringStatus(), PhSensor::stringStatus(), PressureSensor::stringStatus(), and TemperatureSensor::stringStatus().

Here is the caller graph for this function:



#### 4.23.4.8 operator<()

```
bool Sensor::operator< (
    const Sensor & Sensor ) const
```

Operator < overload.

**Parameters**

<a href="#">Sensor</a>	<input type="button" value=""/>
------------------------	---------------------------------

**Returns**

true

false

Definition at line 94 of file Sensor.cpp.

```
94 { return id_ < Sensor.id_; }
```

References id\_.

**4.23.4.9 operator==( )**

```
bool Sensor::operator== (  
    const Sensor & Sensor ) const
```

Operator == overload.

**Parameters**

<a href="#">Sensor</a>	<input type="button" value=""/>
------------------------	---------------------------------

**Returns**

true

false

Definition at line 98 of file Sensor.cpp.

```
98  
99 { return id_ == Sensor.id_;  
100 }
```

References id\_.

**4.23.4.10 operator>( )**

```
bool Sensor::operator> (   
    const Sensor & Sensor ) const
```

Operator > overload.

**Parameters**

<a href="#">Sensor</a>	<input type="button" value=""/>
------------------------	---------------------------------

**Returns**

true  
false

Definition at line 96 of file Sensor.cpp.

```
96 { return id_ > Sensor::id_; }
```

References id\_.

#### 4.23.4.11 printData()

```
void Sensor::printData() const [virtual]
```

Print the data of the [Sensor](#).

Reimplemented in [TemperatureSensor](#), [PressureSensor](#), [PhSensor](#), [LightSensor](#), [HydrometerSensor](#), and [AirQualitySensor](#).

Definition at line 122 of file Sensor.cpp.

```
122 {
123     cout << "This prints the data of sensor " << getType() << " with id" << id_
124     << " please use the correct function to print the data" << endl;
125 // This function will be implemented in the derived classes
126 }
```

Referenced by main().

Here is the caller graph for this function:



#### 4.23.4.12 setData()

```
void Sensor::setData(
    float data)
```

Set the Data object.

**Parameters**

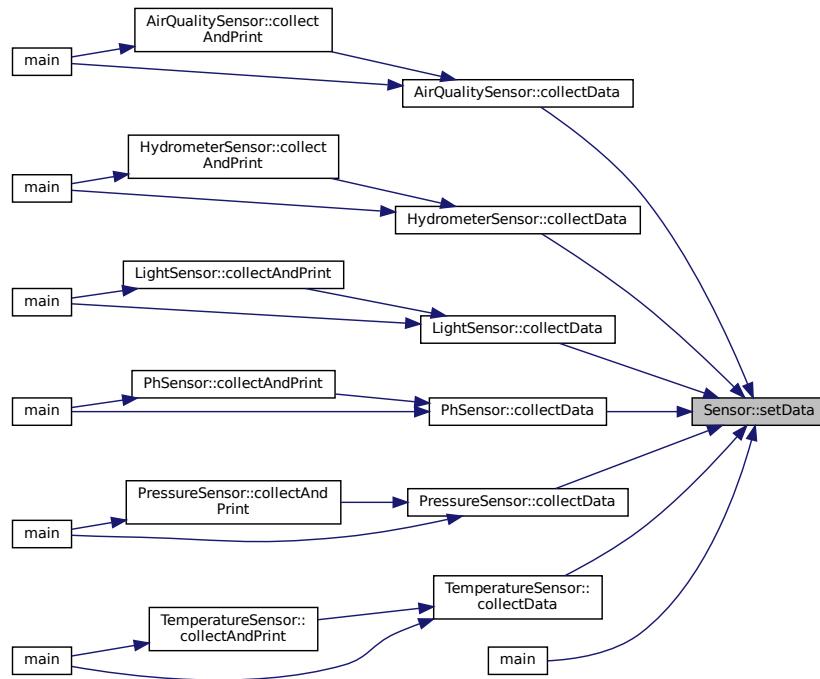
data	
------	--

Definition at line 44 of file Sensor.cpp.

```
44 { data_ = data; }
```

Referenced by AirQualitySensor::collectData(), HydrometerSensor::collectData(), LightSensor::collectData(), PhSensor::collectData(), PressureSensor::collectData(), TemperatureSensor::collectData(), and main().

Here is the caller graph for this function:



#### 4.23.4.13 setId()

```
void Sensor::setId (
    int newid )
```

Set the Id object.

##### Parameters

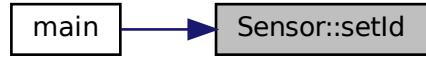
<i>newid</i>	<input type="text"/>
--------------	----------------------

Definition at line 48 of file Sensor.cpp.

```
48 { id_ = newid; }
```

Referenced by main().

Here is the caller graph for this function:



#### 4.23.4.14 setType()

```
void Sensor::setType (  
    std::string newtype )
```

Set the Type object.

##### Parameters

<i>newtype</i>	<input type="text"/>
----------------	----------------------

Definition at line 55 of file Sensor.cpp.  
55 { *type\_* = stringToType(*newtype*); }

Referenced by main().

Here is the caller graph for this function:



#### 4.23.4.15 stringToType()

```
Sensor::Types Sensor::stringToType (   
    const std::string & type ) const [private]
```

Convert the string to the type.

**Parameters**

<i>type</i>	
-------------	--

**Returns****Types**

Definition at line 57 of file Sensor.cpp.

```
57
58     if (type == "TEMPERATURE") {
59         return Types::TEMPERATURE;
60     } else if (type == "AIR_QUALITY") {
61         return Types::AIR_QUALITY;
62     } else if (type == "HYDROMETER") {
63         return Types::HYDROMETER;
64     } else if (type == "PRESSURE") {
65         return Types::PRESSURE;
66     } else if (type == "LIGHT_SENSOR") {
67         return Types::LIGHT_SENSOR;
68     } else if (type == "PH_SENSOR") {
69         return Types::PH_SENSOR;
70     } else {
71         return Types::NONE;
72     }
73 }
```

**4.23.4.16 turnOff()**

```
void Sensor::turnOff ( )
```

Turn off the sensor.

Definition at line 38 of file Sensor.cpp.

```
38 { active_ = false; }
```

Referenced by main().

Here is the caller graph for this function:



#### 4.23.4.17 turnOn()

```
void Sensor::turnOn ( )
```

Turn on the sensor.

Definition at line 40 of file Sensor.cpp.

```
40 { active_ = true; }
```

Referenced by main().

Here is the caller graph for this function:



#### 4.23.4.18 typeToString()

```
std::string Sensor::typeToString (   
    Types type ) const [private]
```

Convert the type to the string.

##### Parameters

type	<input type="text"/>
------	----------------------

##### Returns

`std::string`

Definition at line 75 of file Sensor.cpp.

```
75 {  
76     switch (type) {  
77         case Types::TEMPERATURE:  
78             return "TEMPERATURE";  
79         case Types::AIR_QUALITY:  
80             return "AIR_QUALITY";  
81         case Types::HYDROMETER:  
82             return "HYDROMETER";  
83         case Types::PRESSURE:  
84             return "PRESSURE";  
85         case Types::LIGHT_SENSOR:  
86             return "LIGHT_SENSOR";  
87         case Types::PH_SENSOR:  
88             return "PH_SENSOR";  
89         default:  
90             return "NONE";  
91     }  
92 }
```

## 4.23.5 Friends And Related Function Documentation

### 4.23.5.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const Sensor & Sensor ) [friend]
```

Operator << overload.

#### Parameters

<i>os</i>	
<i>Sensor</i>	

#### Returns

```
std::ostream&
```

Definition at line 102 of file Sensor.cpp.

```
102
103   os << "ID: " << Sensor.getId() << " Type: " << Sensor.getType()
104   << " Active: " << Sensor.isActive() << " Data: " << Sensor.getData()
105   << std::endl;
106   return os;
107 }
```

### 4.23.5.2 operator>>

```
std::istream& operator>> (
    std::istream & is,
    Sensor & Sensor ) [friend]
```

Operator >> overload.

#### Parameters

<i>is</i>	
<i>Sensor</i>	

#### Returns

```
std::istream&
```

Definition at line 109 of file Sensor.cpp.

```
109
110   cout << "Enter sensor ID: ";
111   is >> sensor.id_;
112   cout << "Enter the type: ";
```

```
113     std::string type;
114     is >> type;
115     sensor.setType(type);
116     cout << "Enter sensor active: ";
117     is >> sensor.active_;
118
119     return is;
120 }
```

## 4.23.6 Member Data Documentation

### 4.23.6.1 `active_`

`bool Sensor::active_ [private]`

The state of the sensor.

Definition at line 194 of file Sensor.h.

### 4.23.6.2 `data_`

`float Sensor::data_ [private]`

The data of the sensor.

Definition at line 199 of file Sensor.h.

### 4.23.6.3 `id_`

`int Sensor::id_ [private]`

The id of the sensor.

Definition at line 184 of file Sensor.h.

Referenced by `operator<()`, `operator==()`, and `operator>()`.

### 4.23.6.4 `type_`

`Types Sensor::type_ [private]`

The type of the sensor.

Definition at line 189 of file Sensor.h.

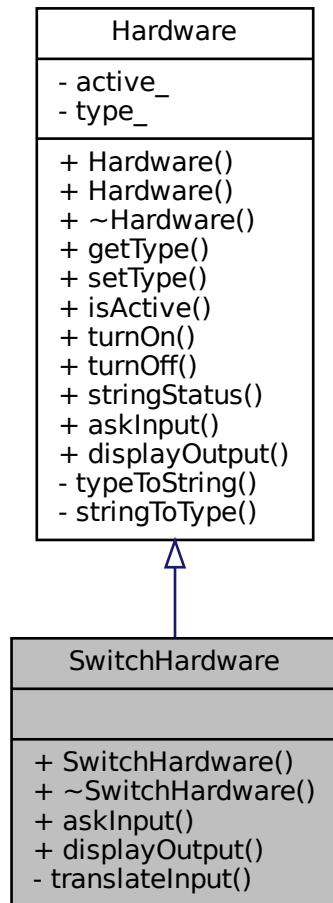
The documentation for this class was generated from the following files:

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

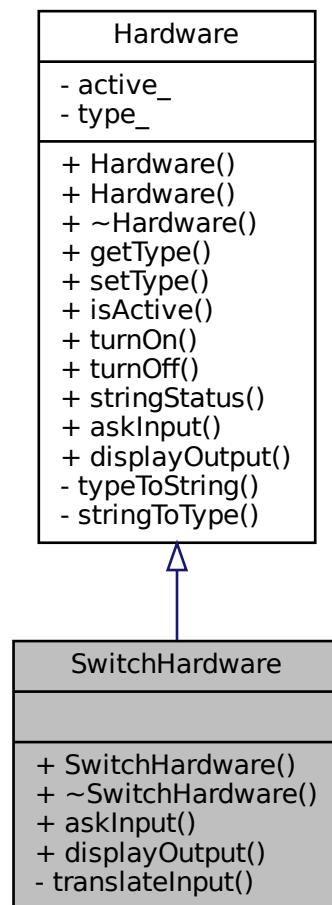
## 4.24 SwitchHardware Class Reference

```
#include <SwitchHardware.h>
```

Inheritance diagram for SwitchHardware:



Collaboration diagram for SwitchHardware:



## Public Member Functions

- `SwitchHardware (bool active)`  
*Construct a new Switch `Hardware` object.*
- `~SwitchHardware () override`  
*Destroy the Switch `Hardware` object.*
- `int askInput () override`  
*Ask for an input.*
- `void displayOutput () const override`  
*Display the output of the Switch `Hardware`.*

## Private Member Functions

- `void translateInput (int input)`  
*The data of the Switch `Hardware`.*

## Additional Inherited Members

### 4.24.1 Detailed Description

Definition at line 14 of file SwitchHardware.h.

### 4.24.2 Constructor & Destructor Documentation

#### 4.24.2.1 SwitchHardware()

```
SwitchHardware::SwitchHardware (
    bool active ) [explicit]
```

Construct a new Switch [Hardware](#) object.

##### Parameters

active	<input type="button" value=""/>
--------	---------------------------------

##### Returns

[SwitchHardware](#) object

Definition at line 7 of file SwitchHardware.cpp.

```
8     : Hardware(active, Hardware::Types_Hardware::SWITCH) {}
```

#### 4.24.2.2 ~SwitchHardware()

```
SwitchHardware::~SwitchHardware ( ) [override]
```

Destroy the Switch [Hardware](#) object.

Definition at line 10 of file SwitchHardware.cpp.

```
10 { }
```

### 4.24.3 Member Function Documentation

### 4.24.3.1 askInput()

```
int SwitchHardware::askInput ( ) [override], [virtual]
```

Ask for an input.

#### Returns

int

Reimplemented from [Hardware](#).

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

```
20     int input;
21     std::string input_string;
22     // Preguntamos al usuario si ON o OFF y luego el input se lo pasamos a
23     // translateInputToBool
24     std::cout << "Switch waiting a input(ON/OFF) ..." << std::endl;
25     std::cin >> input_string;
26     if (input_string == "ON") {
27         input = 1;
28     }
29     if (input_string == "OFF") {
30         input = 0;
31     }
32
33     translateInput(input);
34     return input;
35     // El valor de active_ sera true o false dependiendo de si se activa o se
36     // desactiva el switch
37     // desactiva el switch
38 }
```

References [translateInput\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.24.3.2 `displayOutput()`

```
void SwitchHardware::displayOutput ( ) const [override], [virtual]
```

Display the output of the Switch [Hardware](#).

Reimplemented from [Hardware](#).

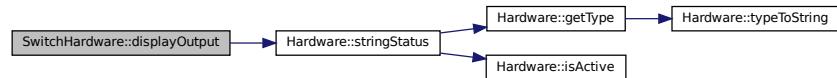
Definition at line 40 of file `SwitchHardware.cpp`.

```
40
41   std::cout << stringStatus() << std::endl;
42 }
```

References `Hardware::stringStatus()`.

Referenced by `main()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.24.3.3 `translateInput()`

```
void SwitchHardware::translateInput (
    int input ) [private]
```

The data of the Switch [Hardware](#).

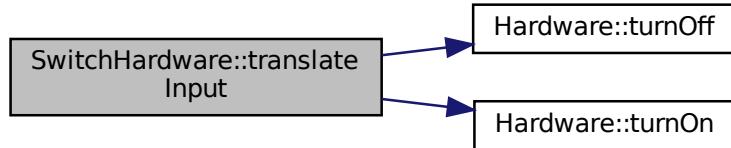
Definition at line 12 of file `SwitchHardware.cpp`.

```
12
13   if (input) {
14     turnOn();
15   } else {
16     turnOff();
17   }
18 }
```

References `Hardware::turnOff()`, and `Hardware::turnOn()`.

Referenced by askInput().

Here is the call graph for this function:



Here is the caller graph for this function:



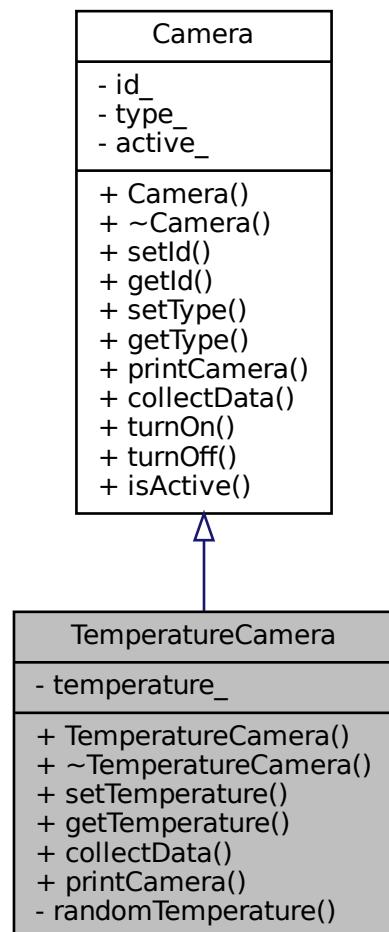
The documentation for this class was generated from the following files:

- src/[SwitchHardware.h](#)
- src/[SwitchHardware.cpp](#)

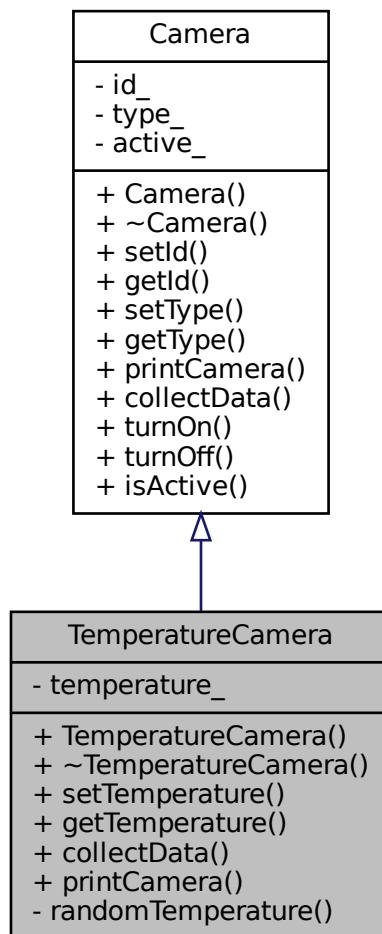
## 4.25 TemperatureCamera Class Reference

```
#include <TemperatureCamera.h>
```

Inheritance diagram for TemperatureCamera:



Collaboration diagram for TemperatureCamera:



## Public Member Functions

- `TemperatureCamera (int id)`  
*Construct a new Temperature Camera object.*
- `~TemperatureCamera ()`  
*Destroy the Temperature Camera object.*
- `void setTemperature (float temperature)`  
*This method sets the temperature of the camera.*
- `float getTemperature ()`  
*This method gets the temperature of the camera.*
- `void collectData () override`  
*This method collects the data of the camera.*
- `void printCamera () override`  
*This method prints the camera information.*

## Private Member Functions

- float [randomTemperature \(\)](#)  
*This method generates a random temperature.*

## Private Attributes

- float [temperature\\_](#)  
*This attribute stores the temperature of the camera.*

### 4.25.1 Detailed Description

Definition at line 19 of file TemperatureCamera.h.

### 4.25.2 Constructor & Destructor Documentation

#### 4.25.2.1 TemperatureCamera()

```
TemperatureCamera::TemperatureCamera (
    int id ) [explicit]
```

Construct a new Temperature [Camera](#) object.

##### Parameters

<i>id</i>	<input type="text"/>
-----------	----------------------

##### Returns

[TemperatureCamera](#) object

Definition at line 10 of file TemperatureCamera.cpp.

```
10 : Camera(id, "TEMPERATURE", true) {
11     temperature_ = -1;
12     std::cout << "Temperature Camera id (" << getId() << ") created" << std::endl;
13 }
```

References Camera::getId(), and temperature\_.

Here is the call graph for this function:



#### 4.25.2.2 ~TemperatureCamera()

```
TemperatureCamera::~TemperatureCamera ( )
```

Destroy the Temperature Camera object.

Definition at line 15 of file TemperatureCamera.cpp.  
15 { }

### 4.25.3 Member Function Documentation

#### 4.25.3.1 collectData()

```
void TemperatureCamera::collectData ( ) [override], [virtual]
```

This method collects the data of the camera.

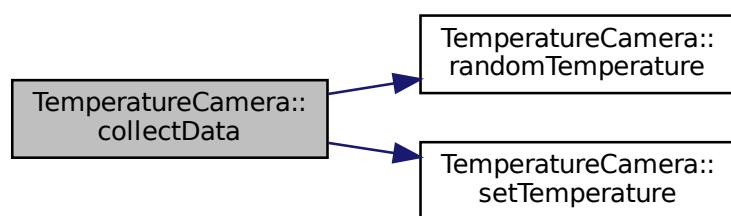
Reimplemented from [Camera](#).

Definition at line 33 of file TemperatureCamera.cpp.  
33 { [setTemperature](#)([randomTemperature](#)()); }

References [randomTemperature\(\)](#), and [setTemperature\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.25.3.2 getTemperature()

```
float TemperatureCamera::getTemperature ( )
```

This method gets the temperature of the camera.

##### Returns

float

Definition at line 21 of file TemperatureCamera.cpp.  
21 { **return temperature\_;** }

References `temperature_`.

Referenced by `ManageCameras::saveCameras()`.

Here is the caller graph for this function:



### 4.25.3.3 printCamera()

```
void TemperatureCamera::printCamera () [override], [virtual]
```

This method prints the camera information.

Reimplemented from [Camera](#).

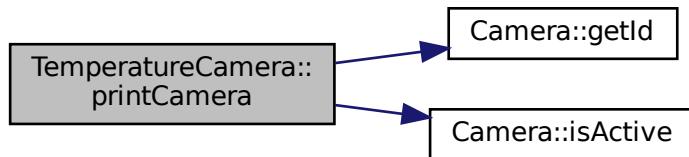
Definition at line 35 of file TemperatureCamera.cpp.

```
35  {
36  if (isActive() and temperature_ != -1) {
37      cout << "Temperature camera (" << getId() << ") record this temperature "
38      << temperature_ << " °C" << endl;
39  } else if (isActive() and temperature_ == -1) {
40      cout << "Temperature camera " << getId()
41      << " is active but has not collected data yet" << endl;
42  } else {
43      cout << "Temperature camera " << getId() << " is not active" << endl;
44  }
45 }
```

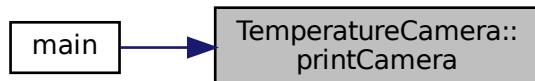
References [Camera::getId\(\)](#), [Camera::isActive\(\)](#), and [temperature\\_](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.25.3.4 randomTemperature()

```
float TemperatureCamera::randomTemperature () [private]
```

This method generates a random temperature.

##### Returns

float

Definition at line 23 of file TemperatureCamera.cpp.

```
23
24 // Random temperature entre 19.0 y 31.0
25 std::random_device rd;
26 std::mt19937 gen(rd());
27 std::uniform_real_distribution<> dis(19.0, 31.0);
28 float temperature = dis(gen);
29 temperature = std::round(temperature * 100) / 100; // Redondea a dos decimales
30 return temperature;
31 }
```

Referenced by `collectData()`.

Here is the caller graph for this function:



#### 4.25.3.5 setTemperature()

```
void TemperatureCamera::setTemperature (
    float temperature )
```

This method sets the temperature of the camera.

##### Parameters

<i>temperature</i>	
--------------------	--

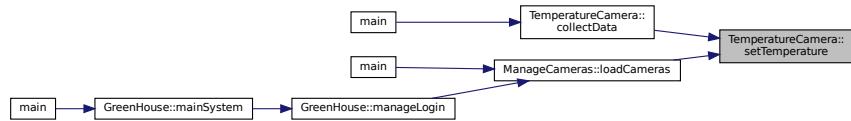
Definition at line 17 of file TemperatureCamera.cpp.

```
17
18     temperature_ = temperature;
19 }
```

References `temperature_`.

Referenced by `collectData()`, and `ManageCameras::loadCameras()`.

Here is the caller graph for this function:



## 4.25.4 Member Data Documentation

### 4.25.4.1 `temperature_`

```
float TemperatureCamera::temperature_ [private]
```

This attribute stores the temperature of the camera.

Definition at line 69 of file TemperatureCamera.h.

Referenced by `getTemperature()`, `printCamera()`, `setTemperature()`, and `TemperatureCamera()`.

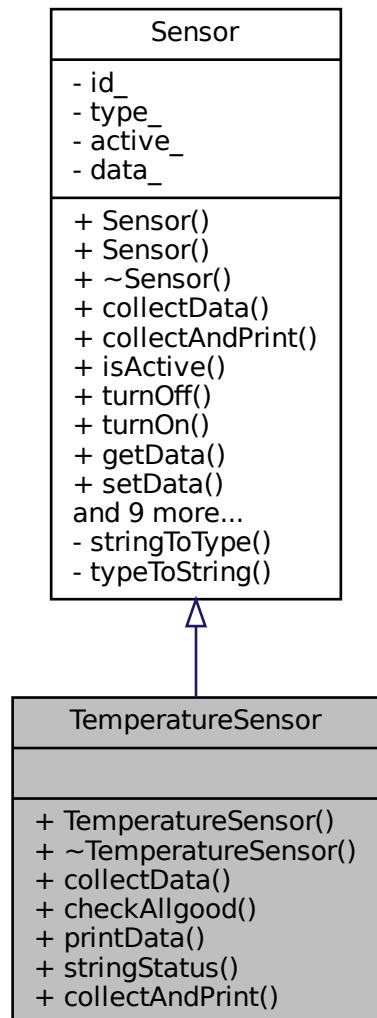
The documentation for this class was generated from the following files:

- src/[TemperatureCamera.h](#)
- src/[TemperatureCamera.cpp](#)

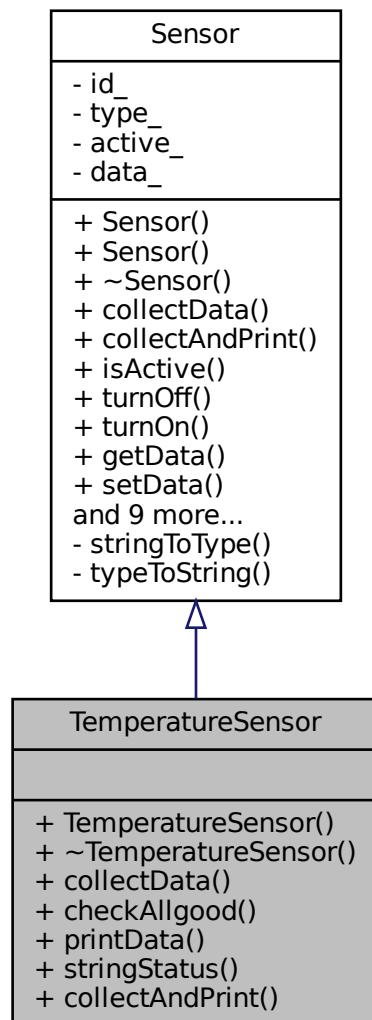
## 4.26 TemperatureSensor Class Reference

```
#include <TemperatureSensor.h>
```

Inheritance diagram for TemperatureSensor:



Collaboration diagram for TemperatureSensor:



## Public Member Functions

- `TemperatureSensor (int id, bool active)`  
*Construct a new Temperature `Sensor` object.*
- `~TemperatureSensor () override`  
*Destroy the Temperature `Sensor` object.*
- `void collectData () override`  
*Collect data of the Temperature `Sensor`.*
- `bool checkAllgood () const override`  
*Check if the Temperature `Sensor` is working properly.*
- `void printData () const override`  
*Print the data of the Temperature `Sensor`.*
- `std::string stringStatus () const`

*String status of the Temperature Sensor.*

- void `collectAndPrint ()`  
*Collect and print the data of the Temperature Sensor.*

## Friends

- std::ostream & `operator<< (std::ostream &os, const TemperatureSensor &sensor)`  
*Overloaded operator<<.*

## Additional Inherited Members

### 4.26.1 Detailed Description

Definition at line 15 of file TemperatureSensor.h.

### 4.26.2 Constructor & Destructor Documentation

#### 4.26.2.1 TemperatureSensor()

```
TemperatureSensor::TemperatureSensor (
    int id,
    bool active ) [explicit]
```

Construct a new Temperature Sensor object.

##### Parameters

<code>id</code>	
<code>active</code>	

##### Returns

TemperatureSensor object

Definition at line 10 of file TemperatureSensor.cpp.

```
11      : Sensor(id, Sensor::Types::TEMPERATURE, active) {}
```

#### 4.26.2.2 ~TemperatureSensor()

```
TemperatureSensor::~TemperatureSensor ( ) [override]
```

Destroy the Temperature Sensor object.

Definition at line 13 of file TemperatureSensor.cpp.

```
13 { }
```

### 4.26.3 Member Function Documentation

#### 4.26.3.1 checkAllgood()

```
bool TemperatureSensor::checkAllgood ( ) const [override], [virtual]
```

Check if the Temperature Sensor is working properly.

##### Returns

true if the Temperature Sensor is working properly  
 false if the Temperature Sensor is not working properly

Reimplemented from [Sensor](#).

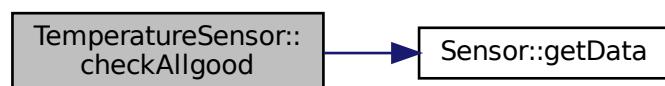
Definition at line 25 of file [TemperatureSensor.cpp](#).

```
25   float data = Sensor::getData();
26   // Entre 20 y 30 estara bien la temperatura, en el resto de los casos no
27   // estara bien
28   if (data >= 20.0f && data <= 30.0f) {
29     return true;
30   } else {
31     return false;
32   }
33 }
```

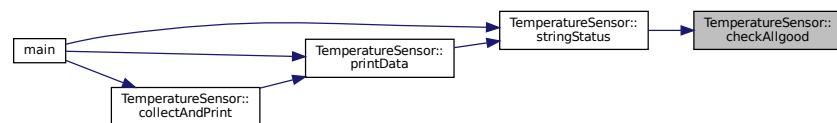
References [Sensor::getData\(\)](#).

Referenced by [stringStatus\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.26.3.2 collectAndPrint()

```
void TemperatureSensor::collectAndPrint ( ) [virtual]
```

Collect and print the data of the Temperature Sensor.

Reimplemented from [Sensor](#).

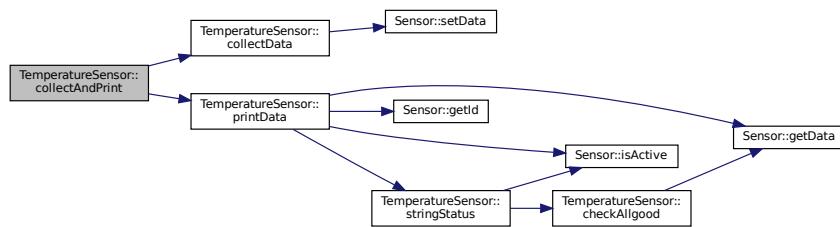
Definition at line 66 of file [TemperatureSensor.cpp](#).

```
66 {  
67     collectData();  
68     printData();  
69 }
```

References [collectData\(\)](#), and [printData\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.26.3.3 collectData()

```
void TemperatureSensor::collectData ( ) [override], [virtual]
```

Collect data of the Temperature Sensor.

This method collects the data of the Temperature Sensor and stores it in the data attribute.

Reimplemented from [Sensor](#).

Definition at line 15 of file TemperatureSensor.cpp.

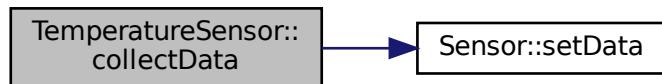
```

15   {
16     // Generamos una temperatura aleatoria entre 19 y 31 grados Centigrados
17
18     std::random_device rd;
19     std::mt19937 gen(rd());
20     std::uniform_real_distribution<> dis(19.0, 31.0);
21     float temperature = dis(gen);
22     Sensor::setData(temperature);
23 }
```

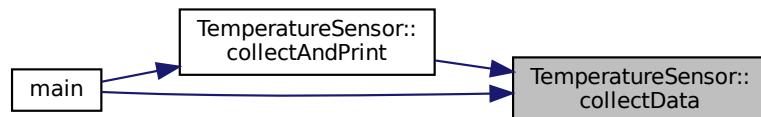
References [Sensor::setData\(\)](#).

Referenced by [collectAndPrint\(\)](#), and [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.26.3.4 [printData\(\)](#)

```
void TemperatureSensor::printData ( ) const [override], [virtual]
```

Print the data of the Temperature [Sensor](#).

Reimplemented from [Sensor](#).

Definition at line 53 of file TemperatureSensor.cpp.

```

53   {
54     // Imprimimos la temperatura actual del sensor, el id que tiene el sensor, y
55     // si todo esta bien o no con (True/False)
56     if (Sensor::isActive\(\)) {
57       std::cout << "Temperature Sensor with "
58       << "ID: " << Sensor::getId\(\) << " - Data: " << Sensor::getData\(\)
59       << " C° - Status: " << stringStatus\(\) << endl;
60   } else {
```

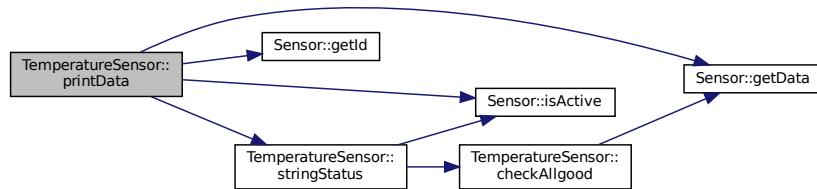
```

61     std::cout << "Temperature Sensor with "
62         << "ID: " << Sensor::getId() << " - INACTIVE" << endl;
63     }
64 }
```

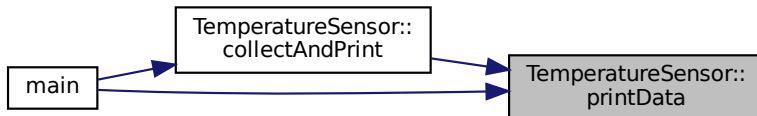
References `Sensor::getData()`, `Sensor::getId()`, `Sensor::isActive()`, and `stringStatus()`.

Referenced by `collectAndPrint()`, and `main()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.26.3.5 stringStatus()

```
std::string TemperatureSensor::stringStatus () const
```

String status of the Temperature [Sensor](#).

Definition at line 41 of file `TemperatureSensor.cpp`.

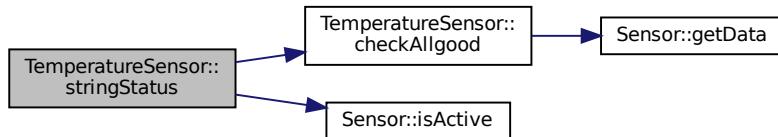
```

41
42     if (Sensor::isActive()) {
43         if (this->checkAllgood()) {
44             return "ACTIVE - GOOD STATUS";
45         } else {
46             return "ACTIVE - BAD STATUS";
47         }
48     } else {
49         return "INACTIVE";
50     }
51 }
```

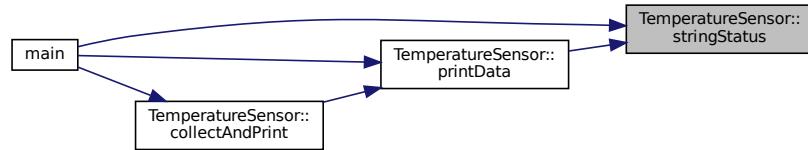
References `checkAllgood()`, and `Sensor::isActive()`.

Referenced by `main()`, and `printData()`.

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.26.4 Friends And Related Function Documentation

### 4.26.4.1 operator<<

```
std::ostream& operator<< (
    std::ostream & os,
    const TemperatureSensor & sensor ) [friend]
```

Overloaded operator<<.

Definition at line 36 of file TemperatureSensor.cpp.

```
36
37     sensor.printData();
38     return os;
39 }
```

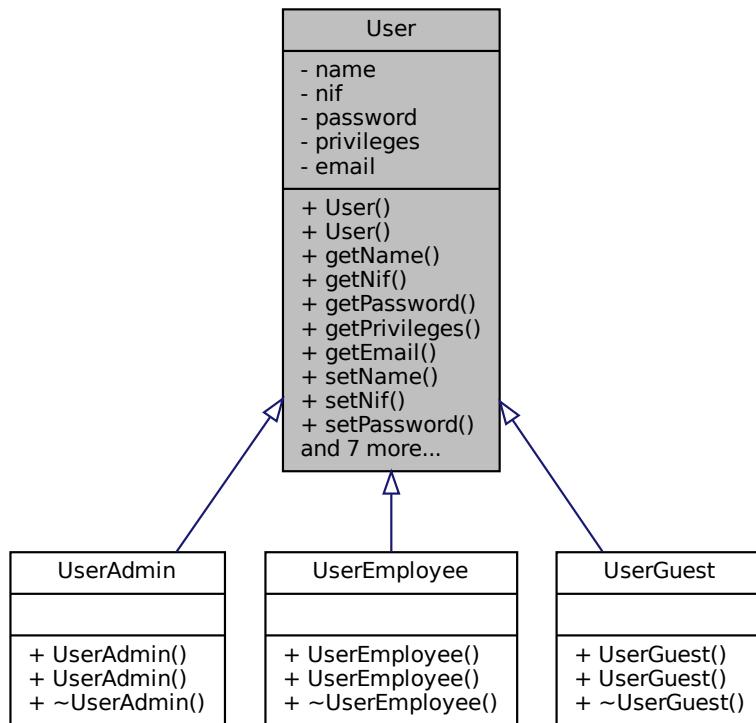
The documentation for this class was generated from the following files:

- [src/TemperatureSensor.h](#)
- [src/TemperatureSensor.cpp](#)

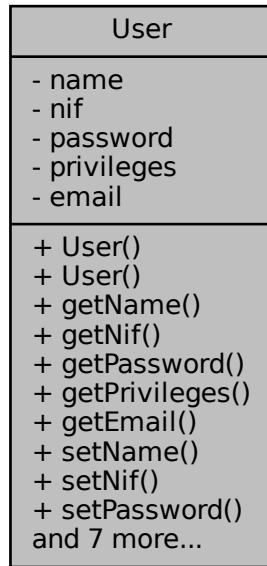
## 4.27 User Class Reference

```
#include <User.h>
```

Inheritance diagram for User:



Collaboration diagram for User:



## Public Member Functions

- **User ()**  
*Construct a new User object.*
- **User (const std::string name, const std::string nif, std::string password, std::string privileges, std::string email)**  
*Construct a new User object.*
- std::string **getName () const**  
*Get the Name object.*
- std::string **getNif () const**  
*Get the Nif object.*
- std::string **getPassword () const**  
*Get the Password object.*
- std::string **getPrivileges () const**  
*Get the Privileges object.*
- std::string **getEmail () const**  
*Get the Email object.*
- void **setName (const std::string name)**  
*Set the Name object.*
- void **setNif (const std::string nif)**  
*Set the Nif object.*
- void **setPassword (const std::string password)**  
*Set the Password object.*
- virtual void **setPrivileges (const std::string privileges)**  
*Set the Privileges object.*
- void **setEmail (const std::string email)**

- Set the `Email` object.
  - bool `operator<` (const `User` &`user`) const
    - Operator < overload (this comparison is made by the privileges)*
  - bool `operator>` (const `User` &`user`) const
    - Operator > overload (this comparison is made by the privileges)*
  - bool `operator==` (const `User` &`user`) const
    - Operator == overload (this comparison is made by the nif)*
  - void `printUser` () const
    - Print the user.*
  - virtual `~User` ()
    - Destroy the `User` object.*

## Private Attributes

- std::string `name`
  - This is the name of the user.*
- std::string `nif`
  - This is the nif of the user.*
- std::string `password`
  - This is the password of the user.*
- std::string `privileges`
  - This is the privileges of the user.*
- std::string `email`
  - This is the email of the user.*

## Friends

- std::ostream & `operator<<` (std::ostream &`os`, const `User` &`user`)
  - Operator << overload.*
- std::istream & `operator>>` (std::istream &`is`, `User` &`user`)
  - Operator >> overload.*

### 4.27.1 Detailed Description

Definition at line 14 of file User.h.

### 4.27.2 Constructor & Destructor Documentation

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

```
User::User ( )
```

Construct a new [User](#) object.

Creates a new [User](#) object with the default values (name, nif, password, privileges, email).

#### Returns

[User](#) object

Definition at line 5 of file User.cpp.

```
5   {
6     name = "";
7     nif = "";
8     password = "";
9     privileges = "";
10    email = "";
11 }
```

References email, name, nif, password, and privileges.

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

```
User::User (
    const std::string name,
    const std::string nif,
    std::string password,
    std::string privileges,
    std::string email ) [explicit]
```

Construct a new [User](#) object.

Creates a new [User](#) object with the values passed as parameters.

#### Parameters

<i>name</i>	of the user
<i>nif</i>	of the user
<i>password</i>	of the user
<i>privileges</i>	of the user
<i>email</i>	of the user

#### Returns

[User](#) object

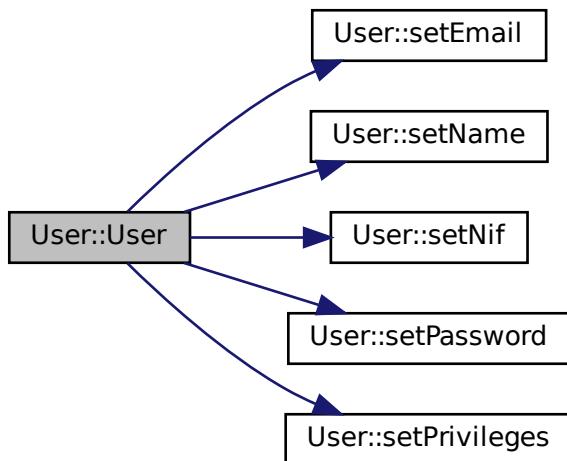
Definition at line 13 of file User.cpp.

```
14   {
15     // LLamar a los metodos set para que se encarguen de hacer las conversiones
16     // necesarias
17     setName(name);
18     setNif(nif);
```

```
19     setPassword(password);
20     setPrivileges(privileges);
21     setEmail(email);
22 }
```

References email, name, nif, password, privileges, setEmail(), setName(), setNif(), setPassword(), and setPrivileges().

Here is the call graph for this function:



#### 4.27.2.3 ~User()

```
User::~User ( ) [virtual]
```

Destroy the [User](#) object.

Definition at line 25 of file [User.cpp](#).  
25 {}

### 4.27.3 Member Function Documentation

#### 4.27.3.1 getEmail()

```
std::string User::getEmail ( ) const
```

Get the Email object.

##### Returns

```
std::string
```

Definition at line 35 of file User.cpp.

```
35 { return email; }
```

References email.

Referenced by main().

Here is the caller graph for this function:



#### 4.27.3.2 getName()

```
std::string User::getName ( ) const
```

Get the Name object.

##### Returns

```
std::string
```

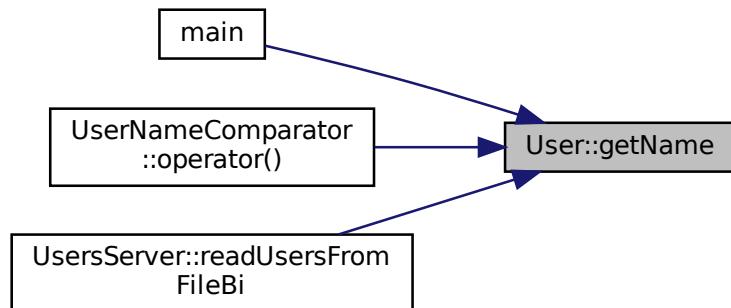
Definition at line 27 of file User.cpp.

```
27 { return name; }
```

References name.

Referenced by main(), UserNameComparator::operator()(), and UsersServer::readUsersFromFileBi().

Here is the caller graph for this function:



#### 4.27.3.3 `getNif()`

`std::string User::getNif ( ) const`

Get the Nif object.

Returns

`std::string`

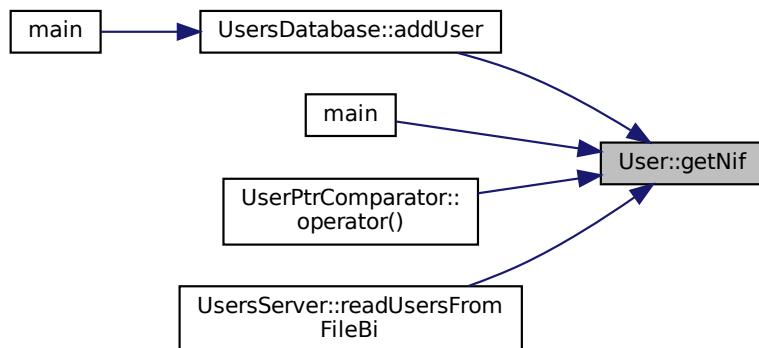
Definition at line 29 of file `User.cpp`.

29 { `return nif;` }

References `nif`.

Referenced by `UsersDatabase::addUser()`, `main()`, `UserPtrComparator::operator()()`, and `UsersServer::readUsersFromFileBi()`.

Here is the caller graph for this function:



#### 4.27.3.4 getPassword()

```
std::string User::getPassword ( ) const
```

Get the Password object.

##### Returns

```
std::string
```

Definition at line 31 of file User.cpp.

```
31 { return password; }
```

References password.

Referenced by main().

Here is the caller graph for this function:



#### 4.27.3.5 getPrivileges()

```
std::string User::getPrivileges ( ) const
```

Get the Privileges object.

##### Returns

```
std::string
```

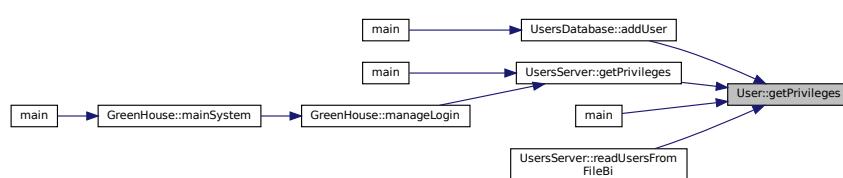
Definition at line 33 of file User.cpp.

```
33 { return privileges; }
```

References privileges.

Referenced by UsersDatabase::addUser(), UsersServer::getPrivileges(), main(), and UsersServer::readUsersFromFileBi().

Here is the caller graph for this function:



#### 4.27.3.6 operator<()

```
bool User::operator< (
    const User & user ) const
```

Operator < overload (this comparison is made by the privileges)

##### Parameters

<i>user</i>	
-------------	--

##### Returns

true

false

Definition at line 62 of file User.cpp.

```
62
63     return privileges > other.privileges;
64 }
```

References privileges.

#### 4.27.3.7 operator==( )

```
bool User::operator== (
    const User & user ) const
```

Operator == overload (this comparison is made by the nif)

##### Parameters

<i>user</i>	
-------------	--

##### Returns

true

false

Definition at line 73 of file User.cpp.

```
73 { return nif == other.nif; }
```

References nif.

#### 4.27.3.8 operator>()

```
bool User::operator> (
    const User & user ) const
```

Operator > overload (this comparison is made by the privileges)

**Parameters**

<i>user</i>	
-------------	--

**Returns**

true  
false

Definition at line 68 of file User.cpp.

```
68
69     return privileges < other.privileges;
70 }
```

References privileges.

**4.27.3.9 printUser()**

```
void User::printUser ( ) const
```

Print the user.

Definition at line 52 of file User.cpp.

```
52
53     std::cout << "-----User-----" << std::endl;
54     std::cout << "Name: " << name << std::endl;
55     std::cout << "NIF: " << nif << std::endl;
56     std::cout << "Password: " << password << std::endl;
57     std::cout << "Privileges: " << privileges << std::endl;
58     std::cout << "Email: " << email << std::endl;
59     std::cout << "-----" << std::endl;
60 }
```

References email, name, nif, password, and privileges.

Referenced by main().

Here is the caller graph for this function:

**4.27.3.10 setEmail()**

```
void User::setEmail (
    const std::string email )
```

Set the Email object.

**Parameters**

<i>email</i>	<input type="text"/>
--------------	----------------------

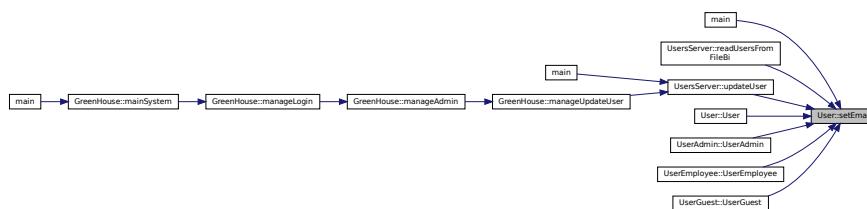
Definition at line 50 of file User.cpp.

```
50 { this->email = email; }
```

References email.

Referenced by main(), UsersServer::readUsersFromFileBi(), UsersServer::updateUser(), User(), UserAdmin::UserAdmin(), UserEmployee::UserEmployee(), and UserGuest::UserGuest().

Here is the caller graph for this function:

**4.27.3.11 setName()**

```
void User::setName (
    const std::string name )
```

Set the Name object.

**Parameters**

<i>name</i>	<input type="text"/>
-------------	----------------------

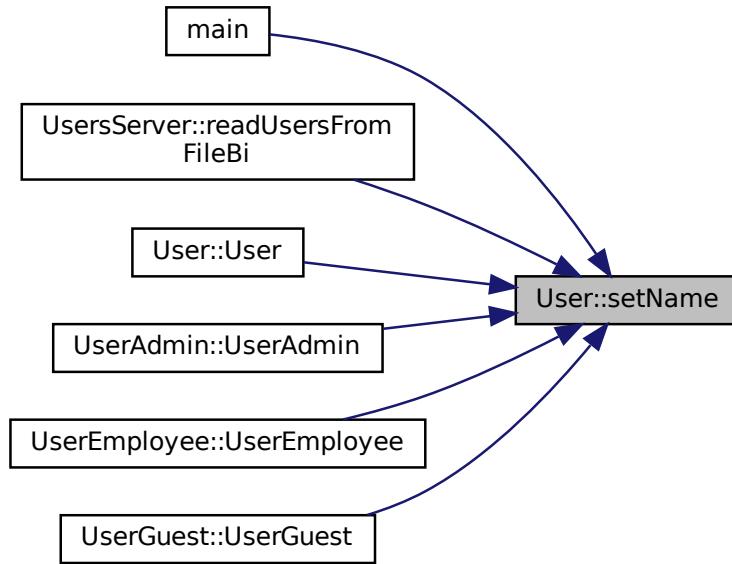
Definition at line 37 of file User.cpp.

```
37 { this->name = name; }
```

References name.

Referenced by main(), UsersServer::readUsersFromFileBi(), User(), UserAdmin::UserAdmin(), UserEmployee::UserEmployee(), and UserGuest::UserGuest().

Here is the caller graph for this function:



#### 4.27.3.12 setNif()

```
void User::setNif ( const std::string nif )
```

Set the Nif object.

##### Parameters

nif	
-----	--

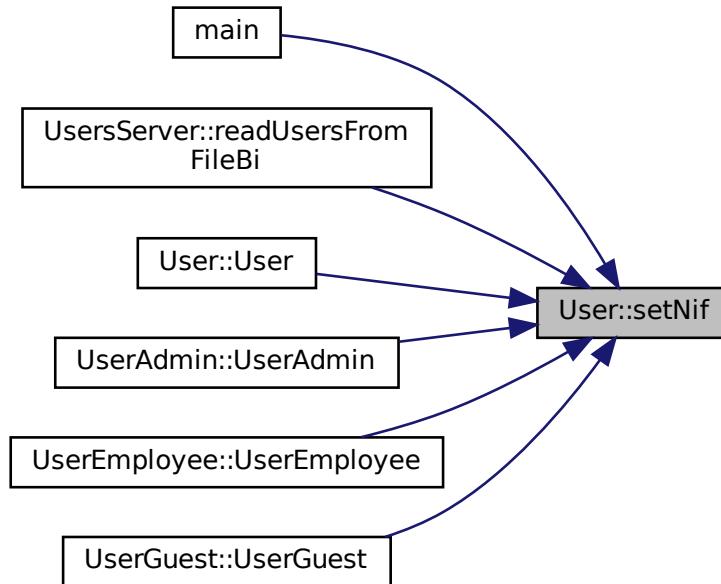
Definition at line 39 of file User.cpp.

```
39 { this->nif = nif; }
```

References nif.

Referenced by main(), UsersServer::readUsersFromFileBi(), User(), UserAdmin::UserAdmin(), UserEmployee::UserEmployee(), and UserGuest::UserGuest().

Here is the caller graph for this function:



#### 4.27.3.13 setPassword()

```
void User::setPassword (
    const std::string password )
```

Set the Password object.

##### Parameters

<code>password</code>	<input type="text"/>
-----------------------	----------------------

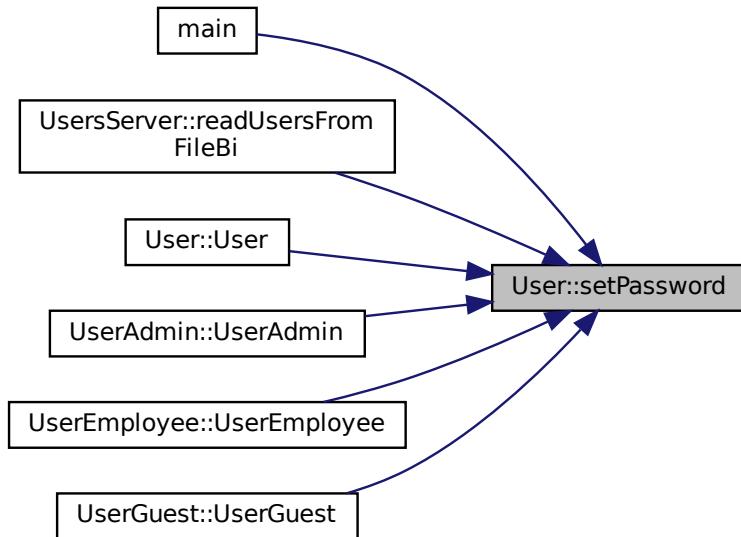
Definition at line 41 of file `User.cpp`.

```
41 { this->password = password; }
```

References `password`.

Referenced by `main()`, `UsersServer::readUsersFromFileBi()`, `User()`, `UserAdmin::UserAdmin()`, `UserEmployee::UserEmployee()`, and `UserGuest::UserGuest()`.

Here is the caller graph for this function:



#### 4.27.3.14 setPrivileges()

```
void User::setPrivileges (
    const std::string privileges ) [virtual]
```

Set the Privileges object.

##### Parameters

<i>privileges</i>	
-------------------	--

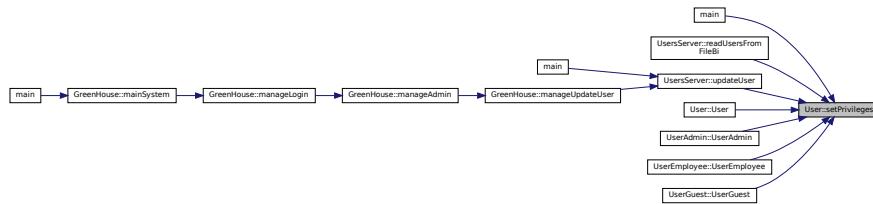
Definition at line 43 of file User.cpp.

```
43
44     for (std::string::size_type i = 0; i < privileges.length(); i++) {
45         privileges[i] = toupper(privileges[i]);
46     }
47     this->privileges = privileges;
48 }
```

References *privileges*.

Referenced by `main()`, `UsersServer::readUsersFromFileBi()`, `UsersServer::updateUser()`, `User()`, `UserAdmin::UserAdmin()`, `UserEmployee::UserEmployee()`, and `UserGuest::UserGuest()`.

Here is the caller graph for this function:



#### 4.27.4 Friends And Related Function Documentation

#### 4.27.4.1 operator<<

```
std::ostream& operator<< ( std::ostream & os, const User & user ) [friend]
```

Operator << overload.

## Parameters

<i>OS</i>	
<i>user</i>	

## Returns

std::ostream&

Definition at line 76 of file User.cpp.

```
76     . . .
77     os << user.getName() << " " << user.getNif() << " " << user.getPassword()
78     << " " << user.getPrivileges() << " " << user.getEmail() << std::endl;
79     return os;
80 }
```

#### 4.27.4.2 operator>>

```
std::istream& operator>> ( std::istream & is,  
                           User & user ) [friend]
```

Operator >> overload.

**Parameters**

<i>is</i>	
<i>user</i>	

**Returns**`std::istream&`

Definition at line 83 of file User.cpp.

```
83
84     std::string privilege;
85     is » user.name » user.nif » user.password » privilege » user.email;
86     user.setPrivileges(privilege);
87     return is;
88 }
```

## 4.27.5 Member Data Documentation

### 4.27.5.1 email

`std::string User::email [private]`

This is the email of the user.

Definition at line 184 of file User.h.

Referenced by `getEmail()`, `printUser()`, `setEmail()`, `User()`, `UserAdmin::UserAdmin()`, `UserEmployee::UserEmployee()`, and `UserGuest::UserGuest()`.

### 4.27.5.2 name

`std::string User::name [private]`

This is the name of the user.

Definition at line 164 of file User.h.

Referenced by `getName()`, `printUser()`, `setName()`, `User()`, `UserAdmin::UserAdmin()`, `UserEmployee::UserEmployee()`, and `UserGuest::UserGuest()`.

#### 4.27.5.3 nif

```
std::string User::nif [private]
```

This is the nif of the user.

Definition at line 169 of file User.h.

Referenced by getNif(), operator==(), printUser(), setNif(), User(), UserAdmin::UserAdmin(), UserEmployee::UserEmployee(), and UserGuest::UserGuest().

#### 4.27.5.4 password

```
std::string User::password [private]
```

This is the password of the user.

Definition at line 174 of file User.h.

Referenced by getPassword(), printUser(), setPassword(), User(), UserAdmin::UserAdmin(), UserEmployee::UserEmployee(), and UserGuest::UserGuest().

#### 4.27.5.5 privileges

```
std::string User::privileges [private]
```

This is the privileges of the user.

Definition at line 179 of file User.h.

Referenced by getPrivileges(), operator<(), operator>(), printUser(), setPrivileges(), and User().

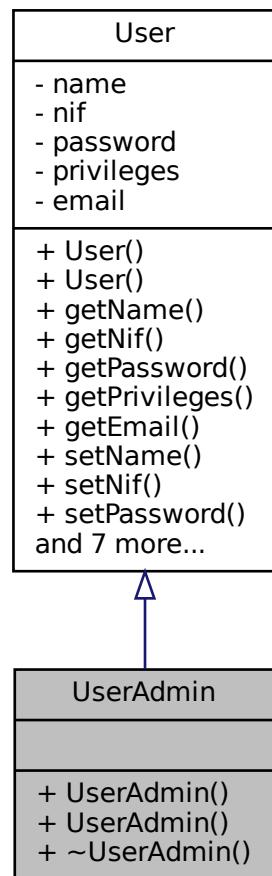
The documentation for this class was generated from the following files:

- src/[User.h](#)
- src/[User.cpp](#)

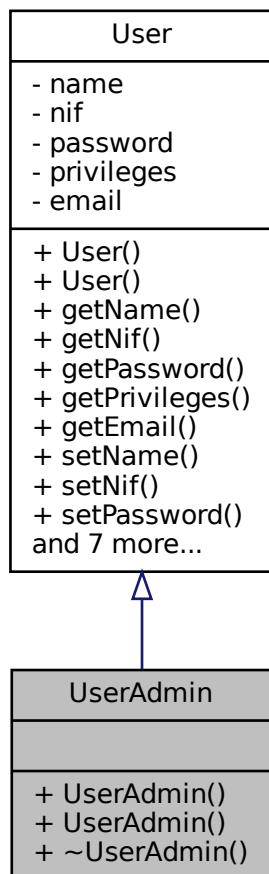
## 4.28 UserAdmin Class Reference

```
#include <UserAdmin.h>
```

Inheritance diagram for UserAdmin:



Collaboration diagram for UserAdmin:



## Public Member Functions

- `UserAdmin ()`  
*Construct a new `User` Admin object.*
- `UserAdmin (const std::string name, const std::string nif, std::string password, std::string email)`  
*Construct a new `User` Admin object.*
- `virtual ~UserAdmin ()`  
*Destroy the `User` Admin object.*

### 4.28.1 Detailed Description

Definition at line 16 of file `UserAdmin.h`.

### 4.28.2 Constructor & Destructor Documentation

### 4.28.2.1 UserAdmin() [1/2]

```
UserAdmin::UserAdmin ( )
```

Construct a new [User Admin](#) object.

Creates a new [UserAdmin](#) object with the default values (name, nif, password, email).

#### Returns

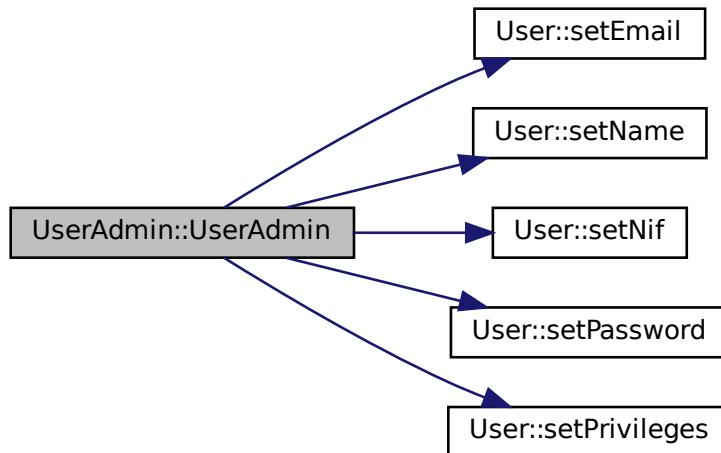
[UserAdmin](#) object

Definition at line 3 of file UserAdmin.cpp.

```
3           {
4     // Utilizar los setters para asignar valores a los atributos
5     setName("");
6     setNif("");
7     setPassword("");
8     setEmail("");
9     setPrivileges("ADMIN");
10 }
```

References [User::setEmail\(\)](#), [User::setName\(\)](#), [User::setNif\(\)](#), [User::setPassword\(\)](#), and [User::setPrivileges\(\)](#).

Here is the call graph for this function:



### 4.28.2.2 UserAdmin() [2/2]

```
UserAdmin::UserAdmin (
    const std::string name,
    const std::string nif,
    std::string password,
    std::string email ) [explicit]
```

Construct a new [User Admin](#) object.

Creates a new [UserAdmin](#) object with the values passed as parameters.

#### Parameters

<i>name</i>	of the user
<i>nif</i>	of the user
<i>password</i>	of the user
<i>email</i>	of the user

#### Returns

[UserAdmin](#) object

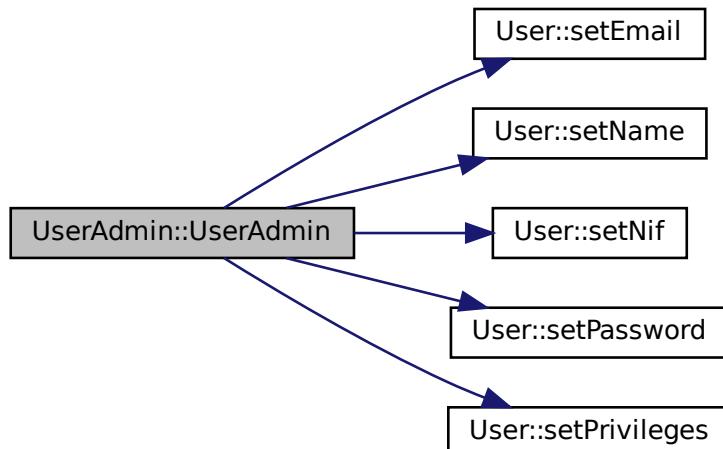
Definition at line 12 of file UserAdmin.cpp.

```

13
14 // Utilizar los setters para asignar valores a los atributos
15 setName(name);
16 setNif(nif);
17 setPassword(password);
18 setEmail(email);
19 setPrivileges("ADMIN");
20 }
```

References `User::email`, `User::name`, `User::nif`, `User::password`, `User::setEmail()`, `User::setName()`, `User::setNif()`, `User::setPassword()`, and `User::setPrivileges()`.

Here is the call graph for this function:



#### 4.28.2.3 ~UserAdmin()

`UserAdmin::~UserAdmin ( ) [virtual]`

Destroy the [User Admin](#) object.

Definition at line 22 of file UserAdmin.cpp.

```
22 {}
```

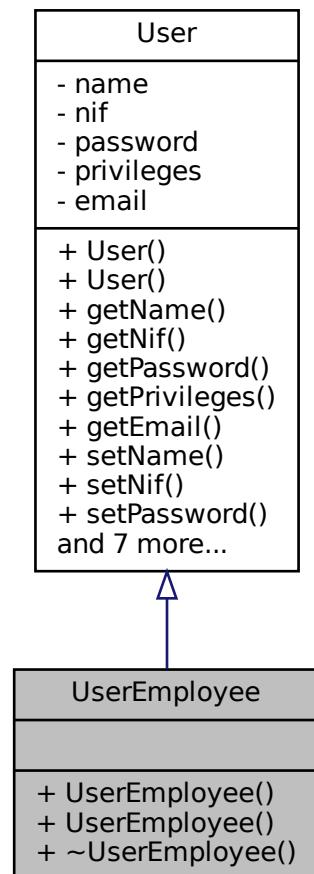
The documentation for this class was generated from the following files:

- [src/UserAdmin.h](#)
- [src/UserAdmin.cpp](#)

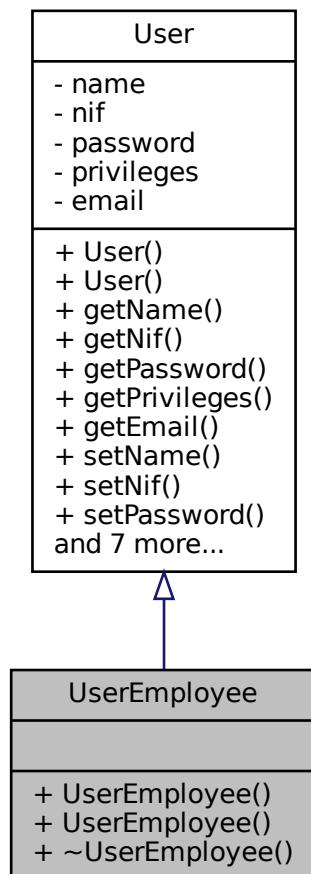
## 4.29 UserEmployee Class Reference

```
#include <UserEmployee.h>
```

Inheritance diagram for UserEmployee:



Collaboration diagram for UserEmployee:



## Public Member Functions

- [UserEmployee \(\)](#)  
*Construct a new `User Employee` object.*
- [UserEmployee \(const std::string `name`, const std::string `nif`, std::string `password`, std::string `email`\)](#)  
*Construct a new `User Employee` object.*
- virtual [~UserEmployee \(\)](#)  
*Destroy the `User Employee` object.*

### 4.29.1 Detailed Description

Definition at line 15 of file `UserEmployee.h`.

### 4.29.2 Constructor & Destructor Documentation

### 4.29.2.1 UserEmployee() [1/2]

```
UserEmployee::UserEmployee ( )
```

Construct a new [User Employee](#) object.

Creates a new [UserEmployee](#) object with the default values (name, nif, password, email).

Returns

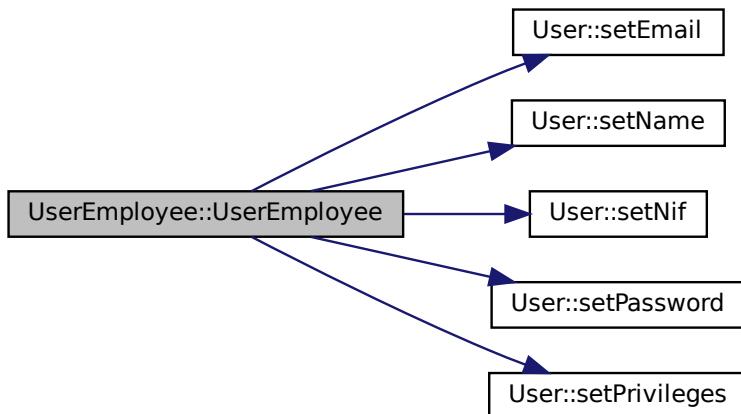
[UserEmployee](#) object

Definition at line 3 of file [UserEmployee.cpp](#).

```
3   {
4     // Utilizar los setters para asignar valores a los atributos
5     setName("");
6     setNif("");
7     setPassword("");
8     setEmail("");
9     setPrivileges("EMPLOYEE");
10 }
```

References [User::setEmail\(\)](#), [User::setName\(\)](#), [User::setNif\(\)](#), [User::setPassword\(\)](#), and [User::setPrivileges\(\)](#).

Here is the call graph for this function:



### 4.29.2.2 UserEmployee() [2/2]

```
UserEmployee::UserEmployee (
    const std::string name,
    const std::string nif,
    std::string password,
    std::string email) [explicit]
```

Construct a new [User Employee](#) object.

Creates a new [UserEmployee](#) object with the values passed as parameters.

### Parameters

<i>name</i>	of the user
<i>nif</i>	of the user
<i>password</i>	of the user
<i>email</i>	of the user

### Returns

[UserEmployee](#) object

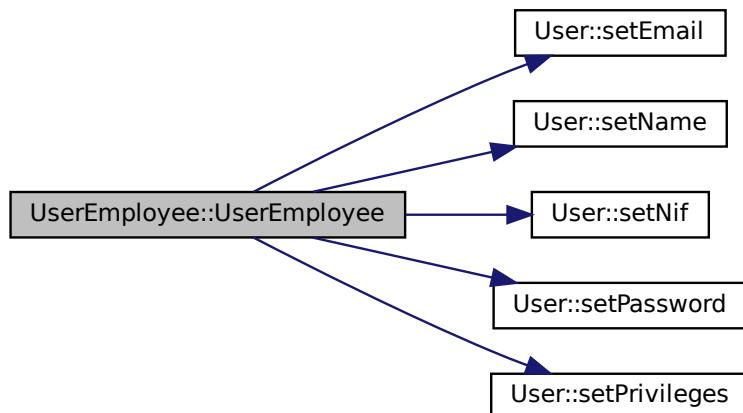
Definition at line 12 of file UserEmployee.cpp.

```

13
14 // Utilizar los setters para asignar valores a los atributos
15 setName(name);
16 setNif(nif);
17 setPassword(password);
18 setEmail(email);
19 setPrivileges("EMPLOYEE");
20 }
```

References `User::email`, `User::name`, `User::nif`, `User::password`, `User::setEmail()`, `User::setName()`, `User::setNif()`, `User::setPassword()`, and `User::setPrivileges()`.

Here is the call graph for this function:



### 4.29.2.3 ~UserEmployee()

`UserEmployee::~UserEmployee () [virtual]`

Destroy the [User Employee](#) object.

Definition at line 22 of file UserEmployee.cpp.

```
22 { }
```

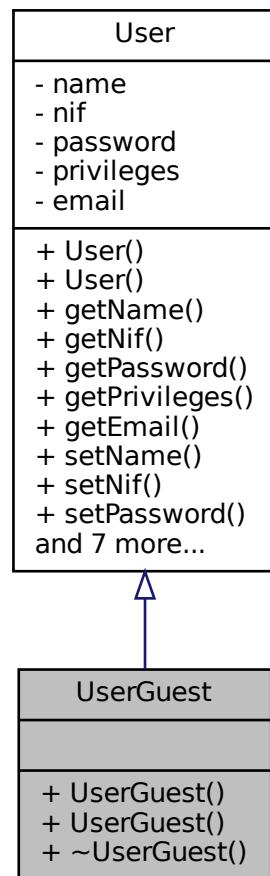
The documentation for this class was generated from the following files:

- [src/UserEmployee.h](#)
- [src/UserEmployee.cpp](#)

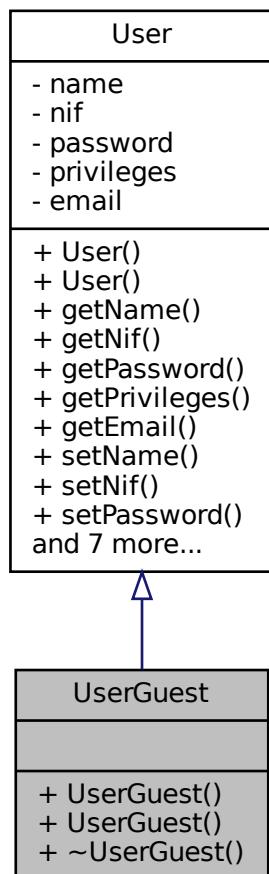
## 4.30 UserGuest Class Reference

```
#include <UserGuest.h>
```

Inheritance diagram for UserGuest:



Collaboration diagram for UserGuest:



## Public Member Functions

- **`UserGuest ()`**  
*Construct a new `User Guest` object.*
- **`UserGuest (const std::string name, const std::string nif, std::string password, std::string email)`**  
*Construct a new `User Guest` object.*
- **`virtual ~UserGuest ()`**  
*Destroy the `User Guest` object.*

### 4.30.1 Detailed Description

Definition at line 16 of file `UserGuest.h`.

### 4.30.2 Constructor & Destructor Documentation

### 4.30.2.1 UserGuest() [1/2]

```
UserGuest::UserGuest ( )
```

Construct a new [User Guest](#) object.

Creates a new [UserGuest](#) object with the default values (name, nif, password, email).

#### Returns

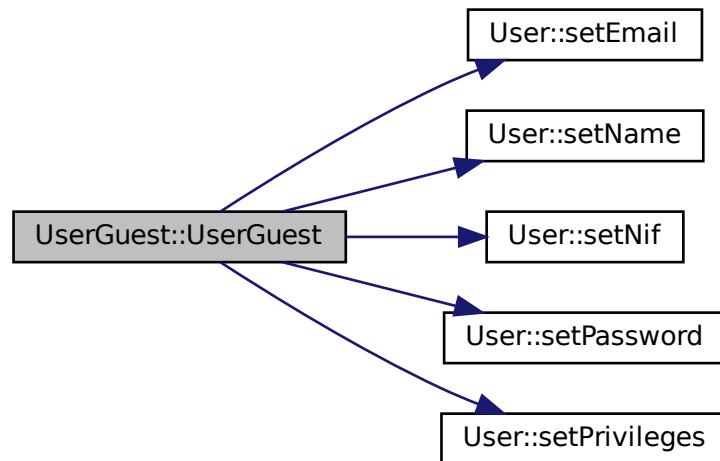
[UserGuest](#) object

Definition at line 3 of file UserGuest.cpp.

```
3   {
4     // Utilizar los setters para asignar valores a los atributos
5     setName("");
6     setNif("");
7     setPassword("");
8     setEmail("");
9     setPrivileges("GUEST");
10 }
```

References [User::setEmail\(\)](#), [User::setName\(\)](#), [User::setNif\(\)](#), [User::setPassword\(\)](#), and [User::setPrivileges\(\)](#).

Here is the call graph for this function:



### 4.30.2.2 UserGuest() [2/2]

```
UserGuest::UserGuest (
    const std::string name,
    const std::string nif,
    std::string password,
    std::string email ) [explicit]
```

Construct a new [User Guest](#) object.

Creates a new [UserGuest](#) object with the values passed as parameters.

#### Parameters

<i>name</i>	of the user
<i>nif</i>	of the user
<i>password</i>	of the user
<i>email</i>	of the user

#### Returns

[UserGuest](#) object

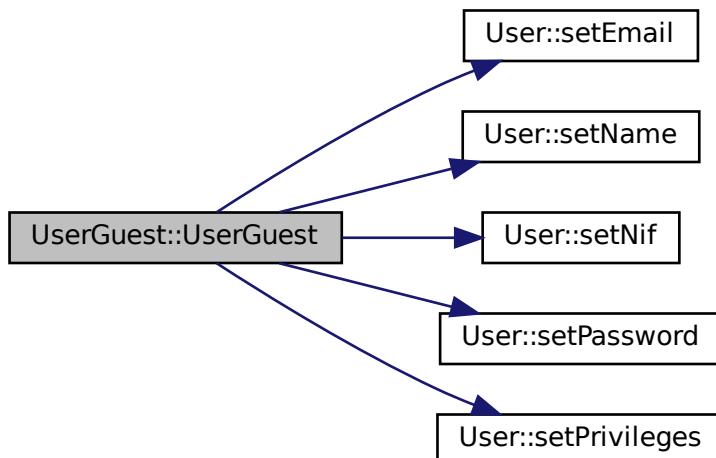
Definition at line 12 of file UserGuest.cpp.

```

13
14 // Utilizar los setters para asignar valores a los atributos
15 setName(name);
16 setNif(nif);
17 setPassword(password);
18 setEmail(email);
19 setPrivileges("GUEST");
20 }
```

References `User::email`, `User::name`, `User::nif`, `User::password`, `User::setEmail()`, `User::setName()`, `User::setNif()`, `User::setPassword()`, and `User::setPrivileges()`.

Here is the call graph for this function:



#### 4.30.2.3 ~UserGuest()

```
UserGuest::~UserGuest ( ) [virtual]
```

Destroy the [User Guest](#) object.

Definition at line 22 of file UserGuest.cpp.

```
22 {}
```

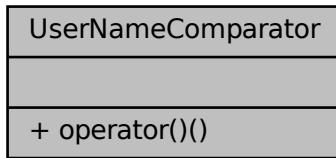
The documentation for this class was generated from the following files:

- [src/UserGuest.h](#)
- [src/UserGuest.cpp](#)

## 4.31 UserNameComparator Class Reference

```
#include <UsersDatabase.h>
```

Collaboration diagram for UserNameComparator:



### Public Member Functions

- bool `operator()` (const `User` \*lhs, const `User` \*rhs) const

#### 4.31.1 Detailed Description

Definition at line 32 of file UsersDatabase.h.

#### 4.31.2 Member Function Documentation

##### 4.31.2.1 operator()

```
bool UserNameComparator::operator() (
    const User * lhs,
    const User * rhs ) const [inline]
```

Definition at line 34 of file UsersDatabase.h.

```
34
35     return lhs->getName() < rhs->getName();
36 }
```

References `User::getName()`.

Here is the call graph for this function:



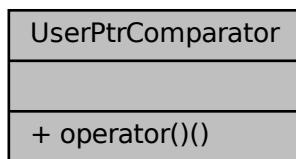
The documentation for this class was generated from the following file:

- src/[UsersDatabase.h](#)

## 4.32 UserPtrComparator Class Reference

```
#include <UsersDatabase.h>
```

Collaboration diagram for UserPtrComparator:



### Public Member Functions

- `bool operator() (const User *us1, const User *us2) const`

#### 4.32.1 Detailed Description

Definition at line 23 of file UsersDatabase.h.

#### 4.32.2 Member Function Documentation

##### 4.32.2.1 operator()

```
bool UserPtrComparator::operator() (
    const User * us1,
    const User * us2 ) const [inline]
```

Definition at line 25 of file UsersDatabase.h.

```
25
26     // Comparar dnis de usuarios si tienen el mismo dni no se puede añadir el
27     // usuario
28     return us1->getNif() < us2->getNif();
29 }
```

References `User::getNif()`.

Here is the call graph for this function:



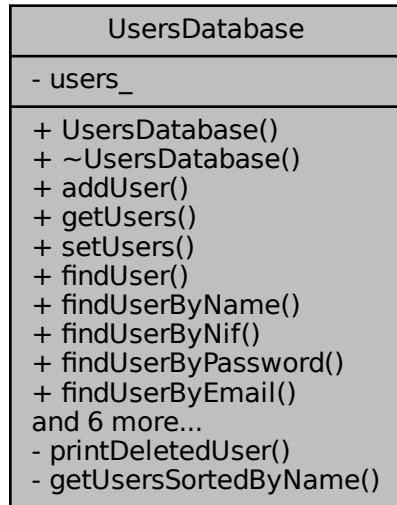
The documentation for this class was generated from the following file:

- [src/UsersDatabase.h](#)

## 4.33 UsersDatabase Class Reference

```
#include <UsersDatabase.h>
```

Collaboration diagram for UsersDatabase:



### Public Member Functions

- [UsersDatabase \(\)](#)

*Construct a new Users Database object* Creates a new `UsersDatabase` object that contains the pointers to users.

- [~UsersDatabase \(\)](#)

- Destroy the Users Database object.*
- void `addUser` (const `User` \*`user`)  
*Add a user to the set of users.*
  - `std::set< const User *, UserPtrComparator > getUsers` () const  
*Get the Users object.*
  - void `setUsers` (const `std::set< const User *, UserPtrComparator >` &`users`)  
*Set the Users object.*
  - `User * findUser` (const `User` &`user`) const  
*Find a user in the set of users.*
  - `User * findUserByName` (const `std::string` `name`) const  
*Find a user in the set of users with the name.*
  - `User * findUserByNif` (const `std::string` `nif`) const  
*Find a user in the set of users with the NIF.*
  - `User * findUserByPassword` (const `std::string` `password`) const  
*Find a user in the set of users with the password.*
  - `User * findUserByEmail` (const `std::string` `email`) const  
*Find a user in the set of users with the email.*
  - void `deleteUser` (const `User` &`user`)  
*Delete a user from the set of users.*
  - void `deleteUserByName` (const `std::string` `name`)  
*Delete a user from the set of users with the name.*
  - void `deleteUserByNif` (const `std::string` `nif`)  
*Delete a user from the set of users with the NIF.*
  - void `deleteUserByEmail` (const `std::string` `email`)  
*Delete a user from the set of users with the email.*
  - bool `isValidPrivileges` (const `std::string` `privileges`) const  
*This method checks if the privileges are valid.*
  - void `printUsers` () const  
*Print all the users.*

## Private Member Functions

- void `printDeletedUser` (const `User` \*`user`) const  
*This method prints the user that has been deleted.*
- `std::set< const User *, UserNameComparator > getUsersSortedByName` () const

## Private Attributes

- `std::set< const User *, UserPtrComparator > users_`  
*This is the set of pointers to users.*

### 4.33.1 Detailed Description

Definition at line 40 of file `UsersDatabase.h`.

### 4.33.2 Constructor & Destructor Documentation

#### 4.33.2.1 UsersDatabase()

```
UsersDatabase::UsersDatabase ( )
```

Construct a new Users Database object Creates a new [UsersDatabase](#) object that contains the pointers to users.

##### Returns

[UsersDatabase](#) object

Definition at line 13 of file [UsersDatabase.cpp](#).

```
13 { }
```

#### 4.33.2.2 ~UsersDatabase()

```
UsersDatabase::~UsersDatabase ( )
```

Destroy the Users Database object.

Definition at line 15 of file [UsersDatabase.cpp](#).

```
15 {  
16     // Liberar la memoria de los usuarios  
17     for (auto user : users_) {  
18         delete user;  
19     }  
20 }
```

References `users_`.

### 4.33.3 Member Function Documentation

#### 4.33.3.1 addUser()

```
void UsersDatabase::addUser (   
    const User * user )
```

Add a user to the set of users.

##### Parameters

<code>user</code>	<input type="text"/>
-------------------	----------------------

Definition at line 28 of file [UsersDatabase.cpp](#).

```
28 {  
29     // Intento añadir un usuario al set de usuarios si no existe, si existe no lo  
30     // añado y llamo a su destructor, tambien tiene que tener unos privilegios  
31     // validos  
32  
33     if (findUserByNif(user->getNif()) == nullptr &&  
34         (isValidPrivileges(user->getPrivileges()))) {
```

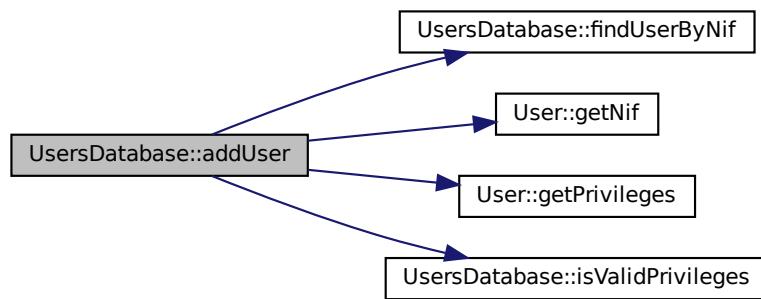
```

35     users_.insert(user);
36 } else {
37     if (!isValidPrivileges(user->getPrivileges())) {
38         std::cout << "Privileges are not valid (ADMIN/EMPLOYEE/GUEST)" 
39                     << std::endl;
40     } else {
41         std::cout << "User already exists" << std::endl;
42     }
43     delete user;
44 }
45 }
```

References `findUserByNif()`, `User::getNif()`, `User::getPrivileges()`, `isValidPrivileges()`, and `users_`.

Referenced by `main()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.33.3.2 `deleteUser()`

```
void UsersDatabase::deleteUser (
    const User & user )
```

Delete a user from the set of users.

**Parameters**

<i>user</i>	
-------------	--

Definition at line 148 of file UsersDatabase.cpp.

```
148
149   for (std::set<const User *>::iterator it = users_.begin(); it != users_.end();
150       it++) {
151     // Si encuentro el usuario lo borro y también el puntero con el destructor
152     // de la clase User
153     if (*it) == user) {
154       printDeletedUser(*it);
155       delete *it; // Llama al destructor de User y libera la memoria
156       users_.erase(it);
157       return;
158     }
159   }
160
161 std::cout << "User not found" << std::endl;
162 std::cout << std::endl;
163 }
```

References `printDeletedUser()`, and `users_`.

Referenced by `main()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.33.3.3 `deleteUserByEmail()`

```
void UsersDatabase::deleteUserByEmail (
    const std::string email )
```

Delete a user from the set of users with the email.

**Parameters**

<i>email</i>	
--------------	--

Definition at line 195 of file UsersDatabase.cpp.

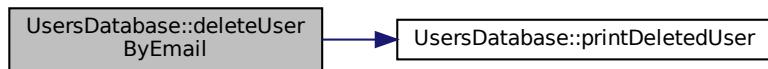
```

195
196     for (std::set<const User *>::iterator it = users_.begin(); it != users_.end();
197         it++) {
198         if ((*it)->getEmail() == email) {
199             printDeletedUser(*it);
200             delete *it; // Llama al destructor de User y libera la memoria
201             users_.erase(it);
202             return;
203         }
204     }
205
206     std::cout << "User not found" << std::endl;
207     std::cout << std::endl;
208 }
```

References **printDeletedUser()**, and **users\_**.

Referenced by **main()**.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.33.3.4 **deleteUserByName()**

```
void UsersDatabase::deleteUserByName (
    const std::string name )
```

Delete a user from the set of users with the name.

## Parameters

<code>name</code>	<input type="text"/>
-------------------	----------------------

Definition at line 165 of file UsersDatabase.cpp.

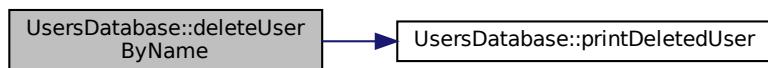
```

165
166   for (std::set<const User *>::iterator it = users_.begin(); it != users_.end();
167       it++) {
168     // Si esta el usuario lo borro y tambien borro el puntero
169     if ((*it)->getName() == name) {
170       printDeletedUser(*it);
171       delete *it; // Llama al destructor de User y libera la memoria
172       users_.erase(it);
173       return;
174     }
175   }
176   std::cout << "User not found" << std::endl;
177   std::cout << std::endl;
178 }
```

References `printDeletedUser()`, and `users_`.

Referenced by `main()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.33.3.5 `deleteUserByNif()`

```

void UsersDatabase::deleteUserByNif (
    const std::string nif )
```

Delete a user from the set of users with the NIF.

**Parameters**

<i>nif</i>	
------------	--

Definition at line 180 of file UsersDatabase.cpp.

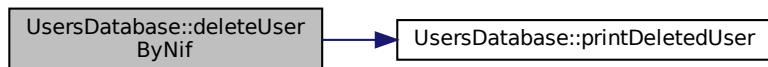
```

180     for (std::set<const User *>::iterator it = users_.begin(); it != users_.end();
181         it++) {
182     if ((*it)->getNif() == nif) {
183         printDeletedUser(*it);
184         delete *it; // Llama al destructor de User y libera la memoria
185         users_.erase(it);
186     }
187     return;
188 }
189
190 std::cout << "User not found" << std::endl;
191 std::cout << std::endl;
192 }
```

References `printDeletedUser()`, and `users_`.

Referenced by `main()`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.33.3.6 `findUser()`

```
User * UsersDatabase::findUser (
    const User & user ) const
```

Find a user in the set of users.

**Parameters**

<i>user</i>	<input type="text"/>
-------------	----------------------

**Returns**

User\*

Definition at line 78 of file UsersDatabase.cpp.

```
78 {  
79     // Si el usuario existe devuelvo el usuario, si no existe devuelvo un usuario  
80     // con todos los atributos vacíos  
81     for (std::set<const User *, UserPtrComparator>::const_iterator it =  
82             users_.begin();  
83             it != users_.end(); it++) {  
84         if (*it) == user) {  
85             return const_cast<User *>(*it);  
86         }  
87     }  
88     return nullptr;  
90 }
```

References users\_.

Referenced by main().

Here is the caller graph for this function:



### 4.33.3.7 findUserByEmail()

```
User * UsersDatabase::findUserByEmail (  
    const std::string email ) const
```

Find a user in the set of users with the email.

**Parameters**

<i>email</i>	<input type="text"/>
--------------	----------------------

**Returns**

User\*

Definition at line 134 of file UsersDatabase.cpp.

```

134
135 // Si el usuario existe devuelvo el usuario, si no existe devuelvo un usuario
136 // con todos los atributos vacios
137 for (std::set<const User *, UserPtrComparator>::const_iterator it =
138     users_.begin();
139     it != users_.end(); it++) {
140     if ((*it)->getEmail() == email) {
141         return const_cast<User *>(*it);
142     }
143 }
144
145 return nullptr;
146 }
```

References users\_.

Referenced by main().

Here is the caller graph for this function:



#### 4.33.3.8 findUserByName()

```
User * UsersDatabase::findUserByName (
    const std::string name ) const
```

Find a user in the set of users with the name.

##### Parameters

<i>name</i>	<input type="text"/>
-------------	----------------------

##### Returns

User\*

Definition at line 92 of file UsersDatabase.cpp.

```

92
93 // Si el usuario existe devuelvo el usuario, si no existe devuelvo un usuario
94 // con todos los atributos vacios
95 for (std::set<const User *, UserPtrComparator>::const_iterator it =
96     users_.begin();
97     it != users_.end(); it++) {
98     if ((*it)->getName() == name) {
99         return const_cast<User *>(*it);
100    }
101 }
```

```
103     return nullptr;
104 }
```

References users\_.

Referenced by main().

Here is the caller graph for this function:



#### 4.33.3.9 findUserByNif()

```
User * UsersDatabase::findUserByNif (
    const std::string nif) const
```

Find a user in the set of users with the NIF.

##### Parameters

nif	
-----	--

##### Returns

User\*

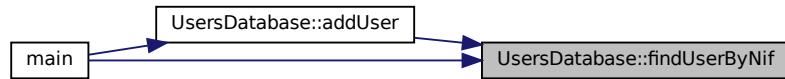
Definition at line 106 of file UsersDatabase.cpp.

```
106
107 // Si el usuario existe devuelvo el usuario, si no existe devuelvo un usuario
108 // con todos los atributos vacios
109 for (std::set<const User *, UserPtrComparator>::const_iterator it =
110       users_.begin();
111       it != users_.end(); it++) {
112     if ((*it)->getNif() == nif) {
113       return const_cast<User *>(*it);
114     }
115   }
116
117 return nullptr;
118 }
```

References users\_.

Referenced by addUser(), and main().

Here is the caller graph for this function:



#### 4.33.3.10 findUserByPassword()

```
User * UsersDatabase::findUserByPassword (
    const std::string password ) const
```

Find a user in the set of users with the password.

##### Parameters

<i>password</i>	<input type="text"/>
-----------------	----------------------

##### Returns

User\*

Definition at line 120 of file UsersDatabase.cpp.

```

120
121     // Si el usuario existe devuelvo el usuario, si no existe devuelvo un usuario
122     // con todos los atributos vacios
123     for (std::set<const User *, UserPtrComparator>::const_iterator it =
124             users_.begin();
125             it != users_.end(); it++) {
126         if ((*it)->getPassword() == password) {
127             return const_cast<User *>(*it);
128         }
129     }
130
131     return nullptr;
132 }
```

References users\_.

#### 4.33.3.11 getUsers()

```
std::set< const User *, UserPtrComparator > UsersDatabase::getUsers ( ) const
```

Get the Users object.

**Returns**

```
std::set<const User *, UserPtrComparator>
```

Definition at line 47 of file UsersDatabase.cpp.

```
47
48     return users_;
49 }
```

References `users_`.

Referenced by `main()`.

Here is the caller graph for this function:



#### 4.33.3.12 getUsersSortedByName()

```
std::set< const User *, UserNameComparator > UsersDatabase::getUsersSortedByName ( ) const
[private]
```

Definition at line 233 of file UsersDatabase.cpp.

```
233
234     std::set<const User *, UserNameComparator> sortedUsers(users_.begin(), users_.end());
235     return sortedUsers;
236 }
```

References `users_`.

#### 4.33.3.13 isValidPrivileges()

```
bool UsersDatabase::isValidPrivileges (
    const std::string privileges ) const
```

This method checks if the privileges are valid.

**Parameters**

<i>privileges</i>	
-------------------	--

**Returns**

true  
false

Definition at line 22 of file UsersDatabase.cpp.

```
22
23 // Compruebo si los privilegios son validos
24 return privileges == "ADMIN" || privileges == "EMPLOYEE" ||
25     privileges == "GUEST";
26 }
```

Referenced by addUser().

Here is the caller graph for this function:

**4.33.3.14 printDeletedUser()**

```
void UsersDatabase::printDeletedUser (
    const User * user ) const [private]
```

This method prints the user that has been deleted.

This method prints the user that has been deleted, is private method because to print a delete user first you have to delete one. So when you delete a user, this method print the user that has been deleted.

**Parameters**

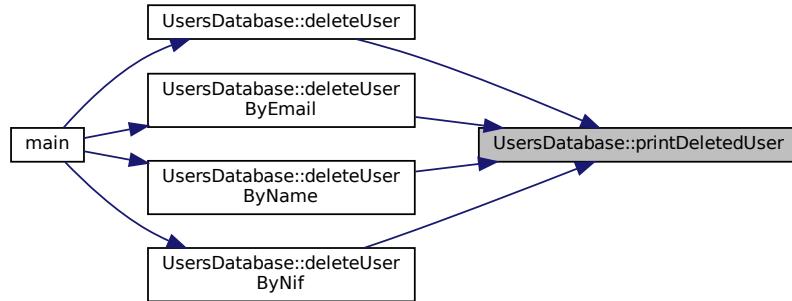
<i>user</i>	<input type="text"/>
-------------	----------------------

Definition at line 227 of file UsersDatabase.cpp.

```
227
228 std::cout << "User deleted: " << (*user).getName() << "-" << (*user).getNif()
229     << std::endl;
230 std::cout << std::endl;
231 }
```

Referenced by deleteUser(), deleteUserByEmail(), deleteUserByName(), and deleteUserByNif().

Here is the caller graph for this function:



#### 4.33.3.15 printUsers()

```
void UsersDatabase::printUsers ( ) const
```

Print all the users.

Definition at line 210 of file UsersDatabase.cpp.

```

210
211     std::cout << "*** LIST OF USERS ***" << std::endl;
212     // Imprimir todos los usuarios
213     /*
214     for (auto it = users_.rbegin(); it != users_.rend(); ++it) {
215         (*it)->printUser();
216         std::cout << std::endl;
217     }
218     */
219     for (const auto &user : users_) {
220         (*user).printUser();
221         std::cout << std::endl;
222     }
223
224     std::cout << "*** END OF LIST ***" << std::endl;
225 }
```

References users\_.

Referenced by main().

Here is the caller graph for this function:



#### 4.33.3.16 setUsers()

```
void UsersDatabase::setUsers (
    const std::set< const User *, UserPtrComparator > & users )
```

Set the Users object.

##### Parameters

<i>users</i>	
--------------	--

Definition at line 51 of file UsersDatabase.cpp.

```
52
53 // El set users debe de convertir los punteros de usuarios a Objetos de
54 // usuarios y luego añadirlos al set de usuarios Bucle para convertirlos en
55 // objetos usuario
56 for (auto user : users) {
57     try {
58         if (user->getPrivileges() == "ADMIN") {
59             users_.insert(new UserAdmin(*static_cast<const UserAdmin *>(user)));
60         } else if (user->getPrivileges() == "EMPLOYEE") {
61             users_.insert(
62                 new UserEmployee(*static_cast<const UserEmployee *>(user)));
63         } else if (user->getPrivileges() == "GUEST") {
64             users_.insert(new UserGuest(*static_cast<const UserGuest *>(user)));
65         } else {
66             std::cerr << "Unknown user privilege: " << user->getPrivileges()
67                 << '\n';
68         }
69     } catch (std::bad_alloc &ba) {
70         std::cerr << "bad_alloc caught: " << ba.what() << '\n';
71         // Borrar el usuario que no se ha podido añadir
72         delete user;
73         throw;
74     }
75 }
76 }
```

References `users_`.

Referenced by `main()`.

Here is the caller graph for this function:



#### 4.33.4 Member Data Documentation

#### 4.33.4.1 `users_`

```
std::set<const User *, UserPtrComparator> UsersDatabase::users_ [private]
```

This is the set of pointers to users.

This is the attribute that contains the pointers to users.

Definition at line 175 of file `UsersDatabase.h`.

Referenced by `addUser()`, `deleteUser()`, `deleteUserByEmail()`, `deleteUserByName()`, `deleteUserByNif()`, `findUser()`, `findUserByEmail()`, `findUserByName()`, `findUserByNif()`, `findUserByPassword()`, `getUsers()`, `getUsersSortedByName()`, `printUsers()`, `setUsers()`, and `~UsersDatabase()`.

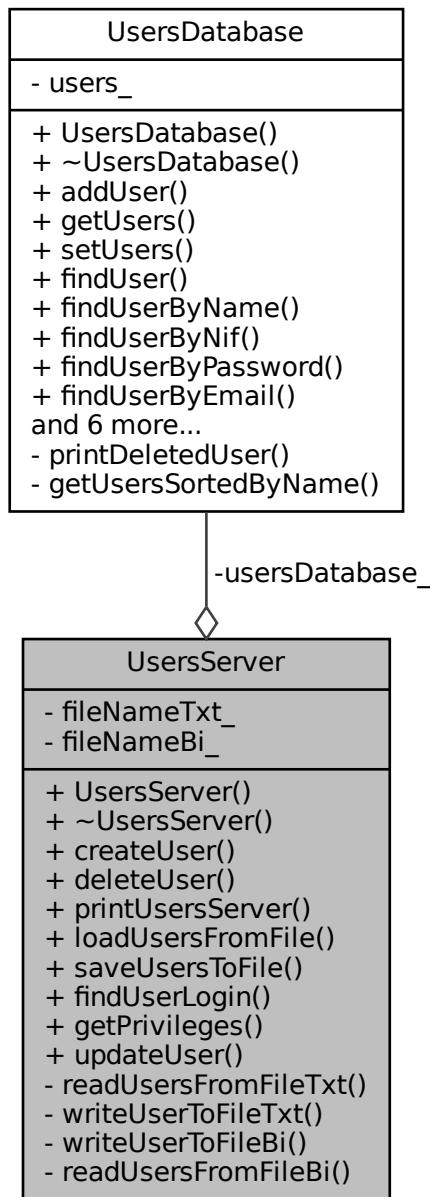
The documentation for this class was generated from the following files:

- src/[UsersDatabase.h](#)
- src/[UsersDatabase.cpp](#)

## 4.34 UsersServer Class Reference

```
#include <UsersServer.h>
```

Collaboration diagram for UsersServer:



## Public Member Functions

- [UsersServer \(\)](#)  
*Construct a new Users Server object* Creates a new `UsersServer` object that contains the `UsersDatabase` object.
- [~UsersServer \(\)](#)  
*Destroy the Users Server object.*
- void [createUser](#) (const std::string name, const std::string nif, const std::string password, const std::string privileges, const std::string email)

- Create a [User](#) object.
  - void [deleteUser](#) (const std::string nif)  
*Delete a User object.*
  - void [printUsersServer](#) () const  
*Print all Users Print all users in the [UsersDatabase](#) object.*
  - void [loadUsersFromFile](#) ()  
*Load Users from File Load users from a file.*
  - void [saveUsersToFile](#) ()  
*Save Users to File Save users to a file.*
- bool [findUserLogin](#) (std::string name, std::string password, std::string nif)  
*Find User Login.*
- std::string [getPrivileges](#) (std::string nif)  
*Get Privileges.*
- void [updateUser](#) (const std::string name, const std::string nif, const std::string password, const std::string privileges, const std::string email)  
*Update User.*

## Private Member Functions

- void [readUsersFromFileTxt](#) ()  
*This method reads the users from a file txt.*
- void [writeUserToFileTxt](#) ()  
*This method writes the users to a file txt.*
- void [writeUserToFileBi](#) ()  
*This method writes the users from a file binary.*
- void [readUsersFromFileBi](#) ()  
*This method reads the users from a file binary.*

## Private Attributes

- [UsersDatabase](#) [usersDatabase\\_](#)  
*This is the [UsersDatabase](#) object.*
- std::string [fileNameTxt\\_](#) = "users.txt"  
*This is the name of the file.*
- std::string [fileNameBi\\_](#) = "users.dat"  
*This is the name of the binary file.*

### 4.34.1 Detailed Description

Definition at line 19 of file UsersServer.h.

### 4.34.2 Constructor & Destructor Documentation

#### 4.34.2.1 UsersServer()

```
UsersServer::UsersServer ( )
```

Construct a new Users Server object Creates a new [UsersServer](#) object that contains the [UsersDatabase](#) object.

##### Returns

[UsersServer](#) object

Definition at line 17 of file [UsersServer.cpp](#).

```
17      {
18      // Creamos tres usuarios por defecto
19      createUser("admin", "12345678X", "admin", "ADMIN", "admin@example.com");
20      createUser("employee", "12345678Y", "employee", "EMPLOYEE",
21                  "employee@example.com");
22      createUser("guest", "12345678Z", "guest", "GUEST", "guest@example.com");
23 }
```

#### 4.34.2.2 ~UsersServer()

```
UsersServer::~UsersServer ( )
```

Destroy the Users Server object.

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

```
25      {
26      // Esto destruira el set de punteros de la base de datos
27 }
```

### 4.34.3 Member Function Documentation

#### 4.34.3.1 createUser()

```
void UsersServer::createUser (
    const std::string name,
    const std::string nif,
    const std::string password,
    const std::string privileges,
    const std::string email )
```

Create a [User](#) object.

##### Parameters

<i>name</i>	
<i>nif</i>	
<i>password</i>	
<i>privileges</i>	
<i>email</i>	

Definition at line 29 of file UsersServer.cpp.

```
32
33 // Debe de crear el usuario y añadir el puntero al set de usuarios
34 User *user = new User(name, nif, password, privileges, email);
35 usersDatabase_.addUser(user);
36 }
```

Referenced by main(), and GreenHouse::manageCreateUser().

Here is the caller graph for this function:



#### 4.34.3.2 deleteUser()

```
void UsersServer::deleteUser (
    const std::string nif )
```

Delete a [User](#) object.

##### Parameters

<i>nif</i>	
------------	--

Definition at line 38 of file UsersServer.cpp.

```
38
39     usersDatabase_.deleteUserByNif(nif);
40 }
```

Referenced by main(), and GreenHouse::manageDeleteUser().

Here is the caller graph for this function:



#### 4.34.3.3 findUserLogin()

```
bool UsersServer::findUserLogin (
    std::string name,
    std::string password,
    std::string nif )
```

Find [User](#) Login.

**Parameters**

<i>name</i>	
<i>password</i>	
<i>nif</i>	

**Returns**

true  
false

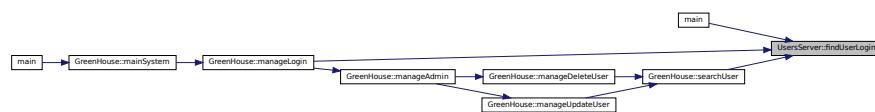
Definition at line 51 of file UsersServer.cpp.

```

52
53     // Debemos de buscar encontrar a un usuario por su nombre, contraseña y nif
54     // Si tienen el mismo puntero devolver true, si no false
55     if (usersDatabase_.findUserByNif(nif) != nullptr) {
56         if (usersDatabase_.findUserByNif(nif)->getName() == name &&
57             usersDatabase_.findUserByNif(nif)->getPassword() == password) {
58             return true;
59         }
60     }
61
62     return false;
63 }
```

Referenced by main(), GreenHouse::manageLogin(), and GreenHouse::searchUser().

Here is the caller graph for this function:

**4.34.3.4 getPrivileges()**

```
std::string UsersServer::getPrivileges (
    std::string nif )
```

Get Privileges.

**Parameters**

<i>nif</i>	
------------	--

**Returns**

std::string the privileges

Definition at line 42 of file UsersServer.cpp.

42

{

```

43     User *user = usersDatabase_.findUserByNif(nif);
44     if (user != nullptr) {
45         return user->getPrivileges();
46     } else {
47         return "GUEST";
48     }
49 }
```

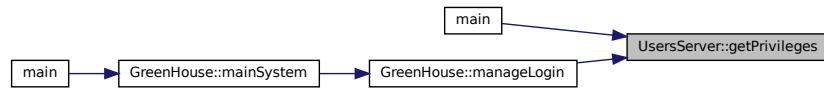
References User::getPrivileges().

Referenced by main(), and GreenHouse::manageLogin().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.34.3.5 loadUsersFromFile()

```
void UsersServer::loadUsersFromFile ( )
```

Load Users from File Load users from a file.

Definition at line 225 of file UsersServer.cpp.

```

225
226     // Cargo los usuarios del archivo
227     // Siguiendo el formato de nombre nif password privilegios email
228
229     try {
230         // readUsersFromFileTxt();
231         readUsersFromFileBi();
232     } catch (FileReadError &e) {
233         std::cerr << e.what() << std::endl;
234
235     } catch (FileCloseError &e) {
236         std::cerr << e.what() << std::endl;
237
238     } catch (FileOpenError &e) {
239         std::cerr << e.what() << std::endl;
240
241     } catch (FileNotFoundException &e) {
242         std::cerr << e.what() << std::endl;
243     }
244
245     /*
```

```

246     if (file.is_open()) {
247         std::string name, nif, password, privileges, email;
248         while (file >> name >> nif >> password >> privileges >> email) {
249             createUser(name, nif, password, privileges, email);
250         }
251         file.close();
252     } else {
253         std::cerr << "Error: Unable to open file for reading." << std::endl;
254     }
255 */
256 }
```

Referenced by main(), and GreenHouse::manageLogin().

Here is the caller graph for this function:



#### 4.34.3.6 printUsersServer()

```
void UsersServer::printUsersServer ( ) const
```

Print all Users Print all users in the [UsersDatabase](#) object.

Definition at line 80 of file [UsersServer.cpp](#).

```

80
81     this->usersDatabase\_.printUsers();
82 }
```

Referenced by main(), and GreenHouse::manageAdmin().

Here is the caller graph for this function:



### 4.34.3.7 readUsersFromFileBi()

```
void UsersServer::readUsersFromFileBi ( ) [private]
```

This method reads the users from a file binary.

Definition at line 84 of file UsersServer.cpp.

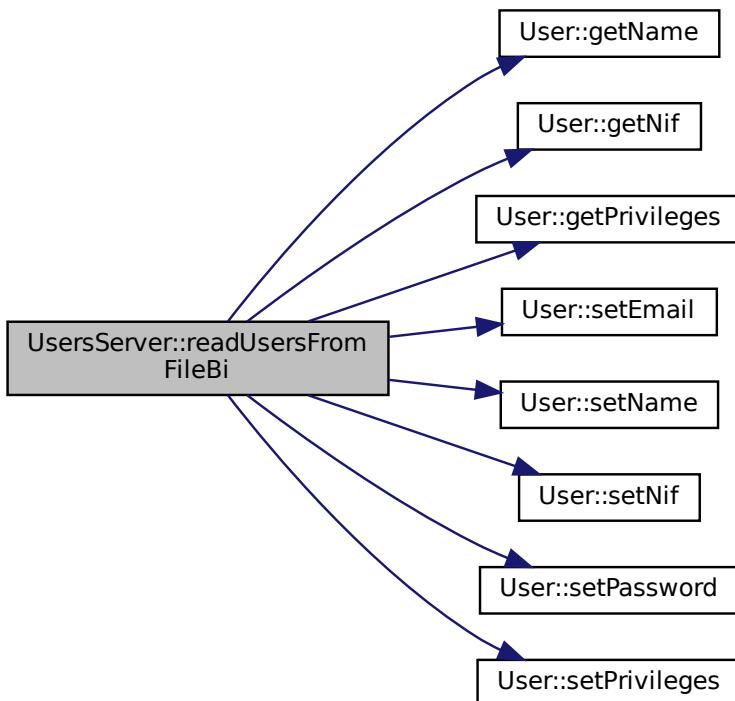
```
84
85     std::cout << "Reading users from file " << fileNameBi_ << "..." << std::endl;
86
87     std::ifstream file(fileNameBi_, std::ios::binary);
88     if (file.is_open()) {
89         while (file) {
90             User *user = new User();
91
92             // Read the user name
93             size_t nameLength;
94             file.read(reinterpret_cast<char *>(&nameLength), sizeof(nameLength));
95             if (!file) {
96                 delete user;
97                 break;
98             }
99             char *name = new char[nameLength];
100            file.read(name, nameLength);
101            if (!file) {
102                delete[] name;
103                delete user;
104                break;
105            }
106            user->setName(std::string(name, nameLength));
107            delete[] name;
108
109            // Read the user nif
110            size_t nifLength;
111            file.read(reinterpret_cast<char *>(&nifLength), sizeof(nifLength));
112            if (!file) {
113                delete user;
114                break;
115            }
116            char *nif = new char[nifLength];
117            file.read(nif, nifLength);
118            if (!file) {
119                delete[] nif;
120                delete user;
121                break;
122            }
123            user->setNif(std::string(nif, nifLength));
124            delete[] nif;
125
126            // Read the user password
127            size_t passwordLength;
128            file.read(reinterpret_cast<char *>(&passwordLength),
129                      sizeof(passwordLength));
130            if (!file) {
131                delete user;
132                break;
133            }
134            char *password = new char[passwordLength];
135            file.read(password, passwordLength);
136            if (!file) {
137                delete[] password;
138                delete user;
139                break;
140            }
141            user->setPassword(std::string(password, passwordLength));
142            delete[] password;
143
144            // Read the user privileges
145            size_t privilegesLength;
146            file.read(reinterpret_cast<char *>(&privilegesLength),
147                      sizeof(privilegesLength));
148            if (!file) {
149                delete user;
150                break;
151            }
152            char *privileges = new char[privilegesLength];
153            file.read(privileges, privilegesLength);
154            if (!file) {
155                delete[] privileges;
156                delete user;
157                break;
158            }
159            user->setPrivileges(std::string(privileges, privilegesLength));
```

```

160     delete[] privileges;
161
162     // Read the user email
163     size_t emailLength;
164     file.read(reinterpret_cast<char *>(&emailLength), sizeof(emailLength));
165     if (!file) {
166         delete user;
167         break;
168     }
169     char *email = new char[emailLength];
170     file.read(email, emailLength);
171     if (!file) {
172         delete[] email;
173         delete user;
174         break;
175     }
176     user->setEmail(std::string(email, emailLength));
177     delete[] email;
178
179     if (file.eof()) {
180         delete user; // prevent adding incomplete user due to eof
181         break;
182     }
183
184     std::cout << "Read user: " << user->getName() << "-" << user->getNif()
185             << "-" << user->getPrivileges() << std::endl;
186     usersDatabase_.addUser(user);
187 }
188 file.close();
189 } else {
190     std::cerr << "Error: Unable to open file for reading." << std::endl;
191 }
192 }
```

References User::getName(), User::getNif(), User::getPrivileges(), User::setEmail(), User::setName(), User::setNif(), User::setPassword(), and User::setPrivileges().

Here is the call graph for this function:



#### 4.34.3.8 `readUsersFromFileTxt()`

```
void UsersServer::readUsersFromFileTxt ( ) [private]
```

This method reads the users from a file txt.

Definition at line 194 of file `UsersServer.cpp`.

```
194      std::cout << "Loading users from file " << fileNameTxt_ << "..." << std::endl;
195  try {
196      std::ifstream file(fileNameTxt_);
197
198      if (!file.good()) {
199          file.close();
200          throw FileNotFoundException(fileNameTxt_);
201      }
202      if (!file.is_open()) {
203          throw FileOpenError(fileNameTxt_);
204      }
205
206      std::string name, nif, password, privileges, email;
207      while (file >> name >> nif >> password >> privileges >> email) {
208          std::cout << "Read user: " << name << " " << nif << " " << password << " "
209          << privileges << " " << email << std::endl;
210          createUser(name, nif, password, privileges, email);
211      }
212
213      file.close();
214
215      if (file.is_open()) {
216          throw FileCloseError(fileNameTxt_);
217      }
218  } catch (std::exception &e) {
219      // std::cerr << e.what() << std::endl;
220      throw;
221  }
222 }
```

#### 4.34.3.9 `saveUsersToFile()`

```
void UsersServer::saveUsersToFile ( )
```

Save Users to File Save users to a file.

Definition at line 341 of file `UsersServer.cpp`.

```
341      {
342      // Guardo los usuarios en el archivo
343
344      try {
345          writeUserToFileTxt();
346          writeUserToFileBi();
347      } catch (FileOpenError &e) {
348          std::cerr << e.what() << std::endl;
349
350      } catch (FileCloseError &e) {
351          std::cerr << e.what() << std::endl;
352
353      } catch (FileWriteError &e) {
354          std::cerr << e.what() << std::endl;
355
356      } catch (FileNotFoundException &e) {
357          std::cerr << e.what() << std::endl;
358      }
359  /*
360  std::cout << "Saving users to file..." << std::endl;
361  std::ofstream file(fileNameTxt_);
362  if (file.is_open()) {
363      for (const User *user : usersDatabase_.getUsers()) {
364          file << user->getName() << " " << user->getNif() << " "
365          << user->getPassword() << " " << user->getPrivileges() << " "
366          << user->getEmail() << std::endl;
367          std::cout << "User saved: " << user->getName() << "-" << user->getNif()
368          << std::endl;
369  }
```

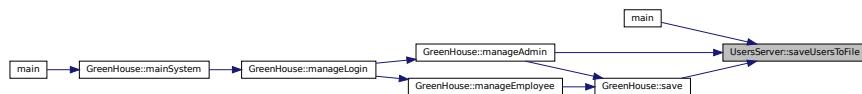
```

370     file.close();
371 } else {
372     std::cerr << "Error: Unable to open file for writing." << std::endl;
373 }
374 */
375 }

```

Referenced by main(), GreenHouse::manageAdmin(), and GreenHouse::save().

Here is the caller graph for this function:



#### 4.34.3.10 updateUser()

```

void UsersServer::updateUser (
    const std::string name,
    const std::string nif,
    const std::string password,
    const std::string privileges,
    const std::string email )

```

Update User.

##### Parameters

<i>name</i>	
<i>nif</i>	
<i>password</i>	
<i>privileges</i>	
<i>email</i>	

Definition at line 65 of file UsersServer.cpp.

```

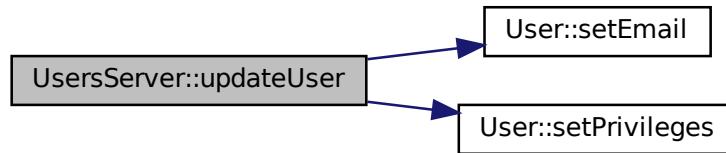
68
69     User *userToUpdate = usersDatabase_.findUserByNif(nif);
70     // Si el usuario existe, y los privilegios que se quieren cambiar
71     // isValidPrivileges entonces se cambian los privilegios y el email
72     if (userToUpdate != nullptr && usersDatabase_.isValidPrivileges(privileges)) {
73         userToUpdate->setPrivileges(privileges);
74         userToUpdate->setEmail(email);
75     } else {
76         std::cout << "User doesn't change, maybe correct privileges" << std::endl;
77     }
78 }

```

References User::setEmail(), and User::setPrivileges().

Referenced by main(), and GreenHouse::manageUpdateUser().

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.34.3.11 writeUserToFileBi()

```
void UsersServer::writeUserToFileBi ( ) [private]
```

This method writes the users from a file binary.

Definition at line 258 of file `UsersServer.cpp`.

```

258     std::cout << "Saving users to file " << fileNameBi_ << "..." << std::endl;
259
260     std::ofstream file(fileNameBi_, std::ios::binary | std::ios::trunc);
261     // Tengo que guardar los usuarios en un archivo binario, recorriendo el set de
262     // usuarios Los guardaremos en el binario siguiendo el formato de nombre nif
263     // password privilegios email y cuando se lea se leera en el mismo formato
264     if (file.is_open()) {
265         for (const User *user : usersDatabase_.getUsers()) {
266
267             // Guardar el nombre
268             size_t nameLength = user->getName().length();
269             file.write(reinterpret_cast<const char *>(&nameLength),
270                         sizeof(nameLength));
271             file.write(user->getName().c_str(), nameLength);
272
273             // Guardar el nif
274             size_t nifLength = user->getNif().length();
275             file.write(reinterpret_cast<const char *>(&nifLength), sizeof(nifLength));
276             file.write(user->getNif().c_str(), nifLength);
277
278             // Guardar el password
279             size_t passwordLength = user->getPassword().length();
280             file.write(reinterpret_cast<const char *>(&passwordLength),
281                         sizeof(passwordLength));
282             file.write(user->getPassword().c_str(), passwordLength);
283
284             // Guardar el privilegios
285             size_t privilegesLength = user->getPrivileges().length();
286             file.write(reinterpret_cast<const char *>(&privilegesLength),
287                         sizeof(privilegesLength));
288             file.write(user->getPrivileges().c_str(), privilegesLength);
289
290             // Guardar el email
  
```

```

292     size_t emailLength = user->getEmail().length();
293     file.write(reinterpret_cast<const char *>(&emailLength),
294                 sizeof(emailLength));
295     file.write(user->getEmail().c_str(), emailLength);
296
297     // Imprimir el usuario que se ha guardado
298     std::cout << "User saved: " << user->getName() << "-" << user->getNif()
299                                     << std::endl;
300 }
301 file.close();
302 } else {
303     std::cerr << "Error: Unable to open file for writing." << std::endl;
304 }
305 }
```

#### 4.34.3.12 writeUserToFileTxt()

void UsersServer::writeUserToFileTxt ( ) [private]

This method writes the users to a file txt.

Definition at line 307 of file UsersServer.cpp.

```

307
308     std::cout << "Saving users to file " << fileNameTxt_ << "..." << std::endl;
309
310     try {
311         std::ofstream file(fileNameTxt_);
312         if (!file.good()) {
313             file.close();
314             throw FileNotFoundException(fileNameTxt_);
315         }
316         if (!file.is_open()) {
317             throw FileOpenError(fileNameTxt_);
318         }
319
320         // Write users to file
321         for (const User *user : usersDatabase_.getUsers()) {
322             file << user->getName() << " " << user->getNif() << " "
323                         << user->getPassword() << " " << user->getPrivileges() << " "
324                         << user->getEmail() << std::endl;
325             std::cout << "User saved: " << user->getName() << "-" << user->getNif()
326                                     << std::endl;
327         }
328
329         // Close the file
330         file.close();
331
332         // Check if file was closed properly
333         if (file.is_open()) {
334             throw FileCloseError(fileNameTxt_);
335         }
336     } catch (std::exception &e) {
337         std::cerr << e.what() << std::endl;
338     }
339 }
```

#### 4.34.4 Member Data Documentation

##### 4.34.4.1 fileNameBi\_

std::string UsersServer::fileNameBi\_ = "users.dat" [private]

This is the name of the binary file.

Definition at line 115 of file UsersServer.h.

#### 4.34.4.2 **fileNameTxt\_**

```
std::string UsersServer::fileNameTxt_ = "users.txt" [private]
```

This is the name of the file.

Definition at line 110 of file [UsersServer.h](#).

#### 4.34.4.3 **usersDatabase\_**

```
UsersDatabase UsersServer::usersDatabase_ [private]
```

This is the [UsersDatabase](#) object.

Definition at line 104 of file [UsersServer.h](#).

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

- src/[UsersServer.h](#)
- src/[UsersServer.cpp](#)

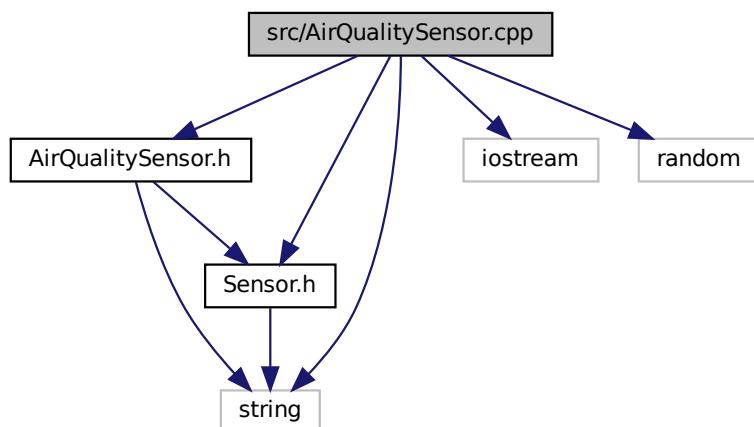


# Chapter 5

## File Documentation

### 5.1 src/AirQualitySensor.cpp File Reference

```
#include "AirQualitySensor.h"
#include <iostream>
#include <random>
#include <string>
#include "Sensor.h"
Include dependency graph for AirQualitySensor.cpp:
```



### Functions

- std::ostream & [operator<<](#) (std::ostream &os, const [AirQualitySensor](#) &sensor)

#### 5.1.1 Function Documentation

### 5.1.1.1 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const AirQualitySensor & sensor )
```

#### Parameters

<i>os</i>	
<i>sensor</i>	

#### Returns

```
std::ostream&
```

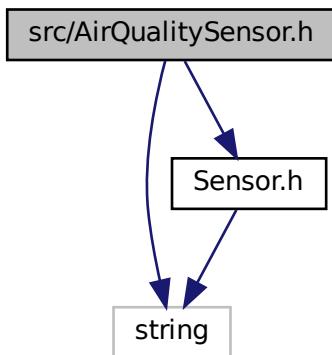
Definition at line 35 of file AirQualitySensor.cpp.

```
35
36     sensor.printData();
37     return os;
38 }
```

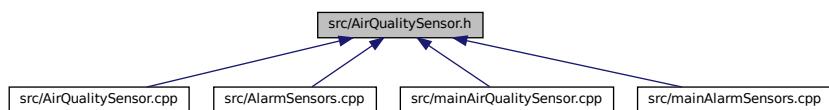
## 5.2 src/AirQualitySensor.h File Reference

This is the class [AirQualitySensor](#). It contains the attributes and methods of the [AirQualitySensor](#) class.

```
#include <string>
#include "Sensor.h"
Include dependency graph for AirQualitySensor.h:
```



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



## Classes

- class [AirQualitySensor](#)

### 5.2.1 Detailed Description

This is the class [AirQualitySensor](#). It contains the attributes and methods of the [AirQualitySensor](#) class.

#### Author

Adrián Montes Linares

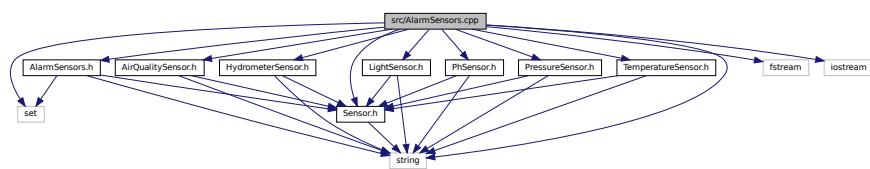
#### Date

21/04/2024

## 5.3 src/AlarmSensors.cpp File Reference

```
#include "AlarmSensors.h"
#include <fstream>
#include <iostream>
#include <set>
#include <string>
#include "Sensor.h"
#include "AirQualitySensor.h"
#include "HydrometerSensor.h"
#include "LightSensor.h"
#include "PhSensor.h"
#include "PressureSensor.h"
#include "TemperatureSensor.h"
```

Include dependency graph for AlarmSensors.cpp:

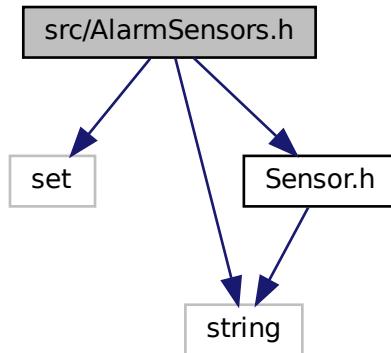


## 5.4 src/AlarmSensors.h File Reference

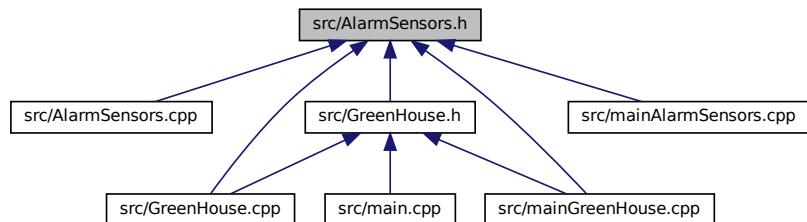
This is the class [AlarmSensors](#). It contains the attributes and methods of the [AlarmSensors](#) class.

```
#include <set>
#include <string>
```

```
#include "Sensor.h"
Include dependency graph for AlarmSensors.h:
```



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



## Classes

- class [AlarmSensors](#)

### 5.4.1 Detailed Description

This is the class [AlarmSensors](#). It contains the attributes and methods of the [AlarmSensors](#) class.

#### Author

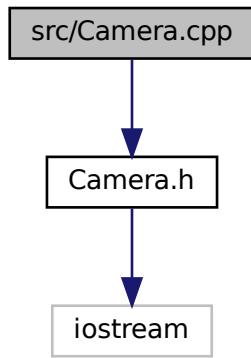
Adrián Montes Linares

#### Date

21/04/2024

## 5.5 src/ Camera.cpp File Reference

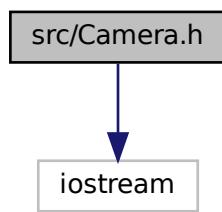
```
#include "Camera.h"  
Include dependency graph for Camera.cpp:
```



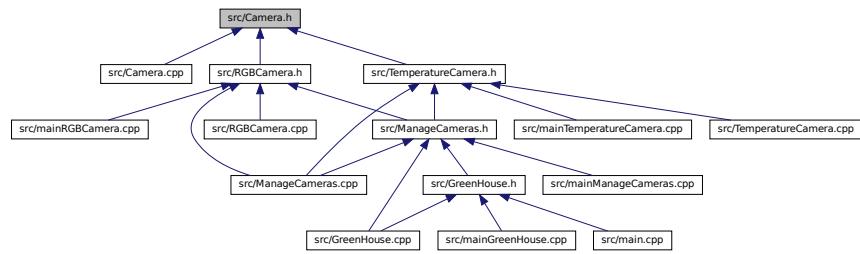
## 5.6 src/ Camera.h File Reference

This is the class [Camera](#). It contains the attributes and methods of the [Camera](#) class. This class is used to represent a camera of the system, it can collect data, turn on/off the camera and print the camera information.

```
#include <iostream>  
Include dependency graph for Camera.h:
```



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



## Classes

- class [Camera](#)

### 5.6.1 Detailed Description

This is the class [Camera](#). It contains the attributes and methods of the [Camera](#) class. This class is used to represent a camera of the system, it can collect data, turn on/off the camera and print the camera information.

#### Author

Adrián Montes Linares

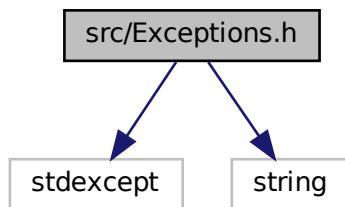
#### Date

23/05/2024

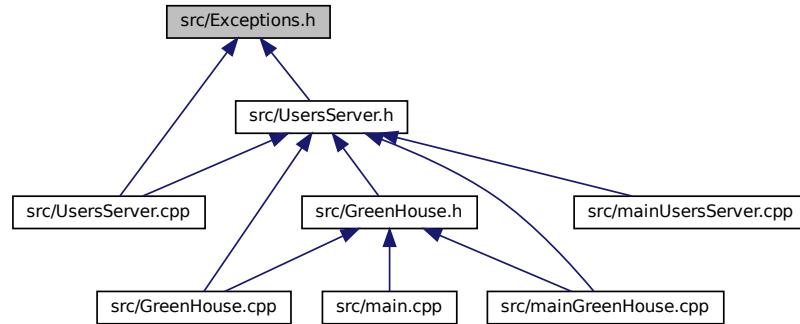
## 5.7 src/Exceptions.h File Reference

This file contains the attributes and methods of the Exceptions class.

```
#include <stdexcept>
#include <string>
Include dependency graph for Exceptions.h:
```



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



## Classes

- class `FileOpenError`
- class `FileCloseError`
- class `FileReadError`
- class `FileWriteError`
- class `FilePermissionError`
- class `FileNotFoundException`
- class `FileLockError`
- class `FileCorruptError`

### 5.7.1 Detailed Description

This file contains the attributes and methods of the Exceptions class.

#### Author

Adrián Montes Linares

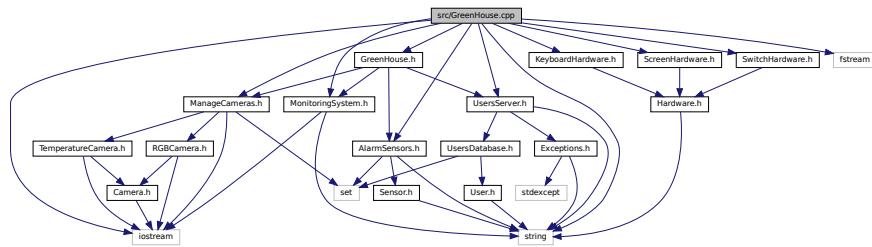
#### Date

21/04/2024

## 5.8 src/GreenHouse.cpp File Reference

```
#include "GreenHouse.h"
#include <fstream>
#include <iostream>
#include <string>
#include "AlarmSensors.h"
#include "KeyboardHardware.h"
#include "ManageCameras.h"
#include "MonitoringSystem.h"
```

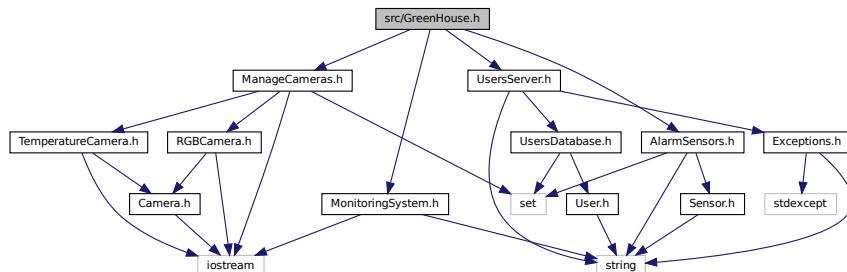
```
#include "ScreenHardware.h"
#include "SwitchHardware.h"
#include "UsersServer.h"
Include dependency graph for GreenHouse.cpp:
```



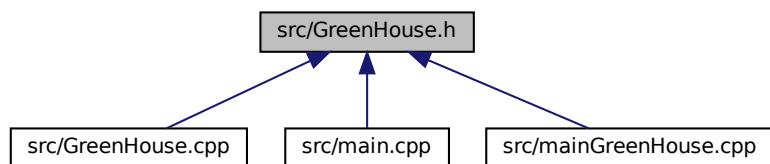
## 5.9 src/GreenHouse.h File Reference

This is the class [GreenHouse](#). It contains the attributes and methods of the [GreenHouse](#) class, this class is the main of the hole system.

```
#include "AlarmSensors.h"
#include "ManageCameras.h"
#include "MonitoringSystem.h"
#include "UsersServer.h"
Include dependency graph for GreenHouse.h:
```



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



## Classes

- class [GreenHouse](#)

### 5.9.1 Detailed Description

This is the class [GreenHouse](#). It contains the attributes and methods of the [GreenHouse](#) class, this class is the main of the hole system.

#### Author

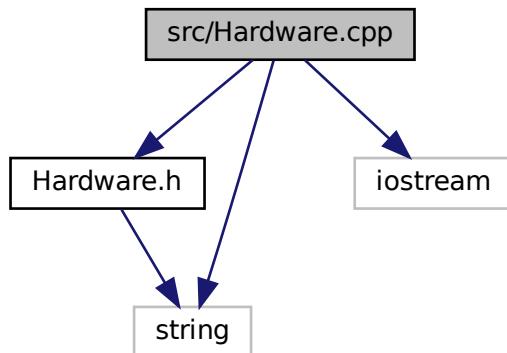
Adrián Montes Linares

#### Date

21/04/2024

## 5.10 src/Hardware.cpp File Reference

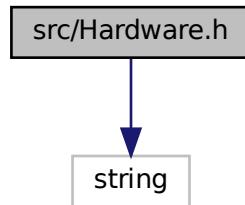
```
#include "Hardware.h"
#include <iostream>
#include <string>
Include dependency graph for Hardware.cpp:
```



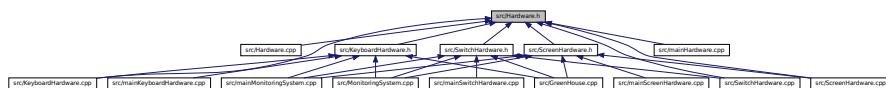
## 5.11 src/Hardware.h File Reference

This is the class [Hardware](#). It contains the attributes and methods of the [Hardware](#) class, this class is the parent of the hole hardware system.

```
#include <string>
Include dependency graph for Hardware.h:
```



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



## Classes

- class [Hardware](#)

### 5.11.1 Detailed Description

This is the class [Hardware](#). It contains the attributes and methods of the [Hardware](#) class, this class is the parent of the hole hardware system.

#### Author

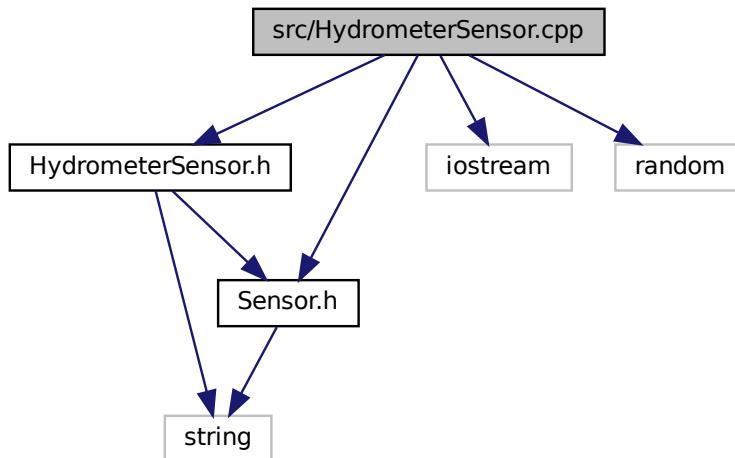
Adrián Montes Linares

#### Date

21/04/2024

## 5.12 src/HydrometerSensor.cpp File Reference

```
#include "HydrometerSensor.h"
#include <iostream>
#include <random>
#include "Sensor.h"
Include dependency graph for HydrometerSensor.cpp:
```



### Functions

- std::ostream & [operator<<](#) (std::ostream &os, const [HydrometerSensor](#) &sensor)

#### 5.12.1 Function Documentation

##### 5.12.1.1 [operator<<\(\)](#)

```
std::ostream& operator<< (
    std::ostream & os,
    const HydrometerSensor & sensor )
```

##### Returns

double

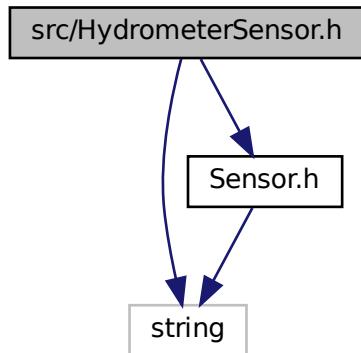
Definition at line 35 of file HydrometerSensor.cpp.

```
35
36     sensor.printData();
37     return os;
38 }
```

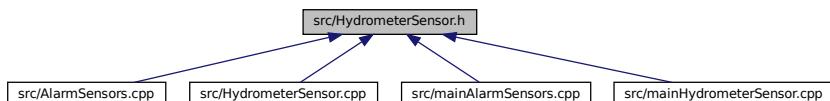
## 5.13 src/HydrometerSensor.h File Reference

This is the class [HydrometerSensor](#). It contains the attributes and methods of the [HydrometerSensor](#) class.

```
#include <string>
#include "Sensor.h"
Include dependency graph for HydrometerSensor.h:
```



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



### Classes

- class [HydrometerSensor](#)

#### 5.13.1 Detailed Description

This is the class [HydrometerSensor](#). It contains the attributes and methods of the [HydrometerSensor](#) class.

##### Author

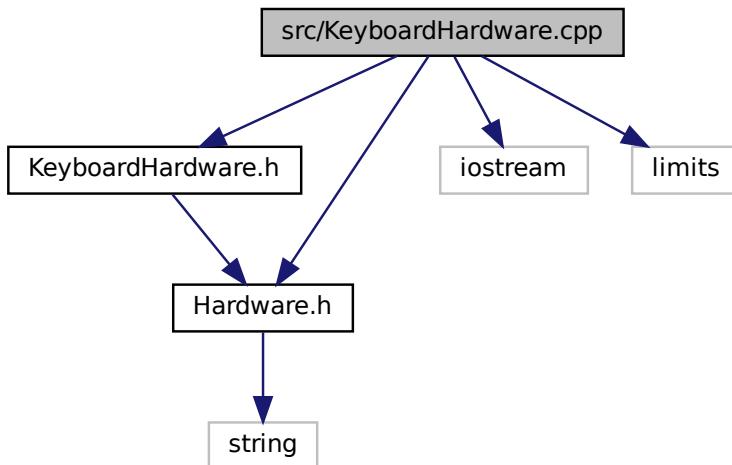
Adrián Montes Linares

##### Date

21/04/2024

## 5.14 src/KeyboardHardware.cpp File Reference

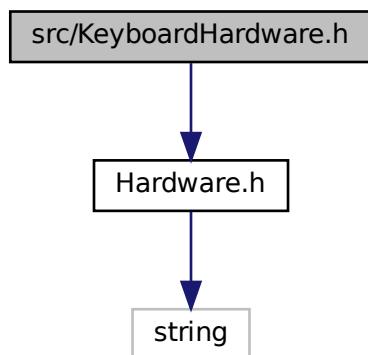
```
#include "KeyboardHardware.h"
#include <iostream>
#include <limits>
#include "Hardware.h"
Include dependency graph for KeyboardHardware.cpp:
```



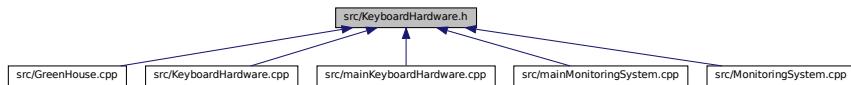
## 5.15 src/KeyboardHardware.h File Reference

This is the class [KeyboardHardware](#). It contains the attributes and methods of the [KeyboardHardware](#) class, this class is a child of the [Hardware](#) class.

```
#include "Hardware.h"
Include dependency graph for KeyboardHardware.h:
```



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



## Classes

- class [KeyboardHardware](#)

### 5.15.1 Detailed Description

This is the class [KeyboardHardware](#). It contains the attributes and methods of the [KeyboardHardware](#) class, this class is a child of the [Hardware](#) class.

#### Author

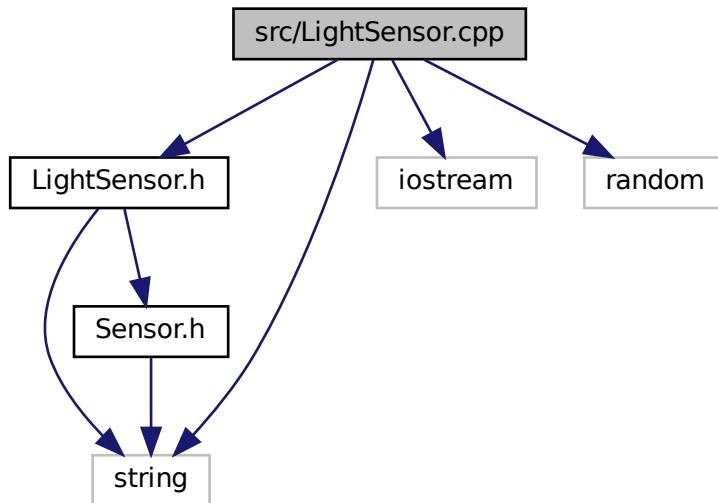
Adrián Montes Linares

#### Date

21/04/2024

## 5.16 src/LightSensor.cpp File Reference

```
#include "LightSensor.h"
#include <iostream>
#include <random>
#include <string>
Include dependency graph for LightSensor.cpp:
```



## Functions

- std::ostream & [operator<<](#) (std::ostream &os, const [LightSensor](#) &sensor)

### 5.16.1 Function Documentation

#### 5.16.1.1 [operator<<\(\)](#)

```
std::ostream& operator<< (
    std::ostream & os,
    const LightSensor & sensor )
```

##### Returns

int

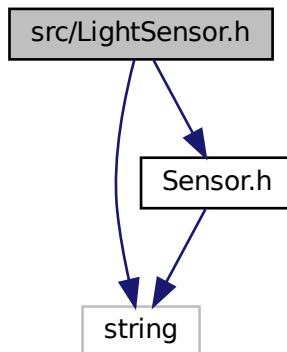
Definition at line 33 of file LightSensor.cpp.

```
33     {
34     sensor.printData();
35     return os;
36 }
```

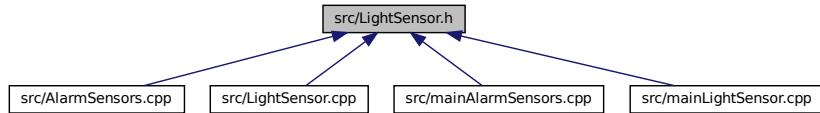
## 5.17 src/LightSensor.h File Reference

This is the class [LightSensor](#). It contains the attributes and methods of the [LightSensor](#) class.

```
#include <string>
#include "Sensor.h"
Include dependency graph for LightSensor.h:
```



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



## Classes

- class [LightSensor](#)

### 5.17.1 Detailed Description

This is the class [LightSensor](#). It contains the attributes and methods of the [LightSensor](#) class.

#### Author

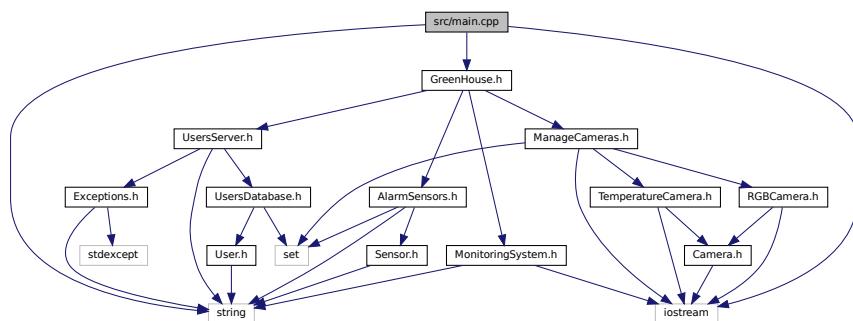
Adrián Montes Linares

#### Date

21/04/2024

## 5.18 src/main.cpp File Reference

```
#include <iostream>
#include <string>
#include "GreenHouse.h"
Include dependency graph for main.cpp:
```



## Functions

- int [main \(\)](#)

## 5.18.1 Function Documentation

### 5.18.1.1 main()

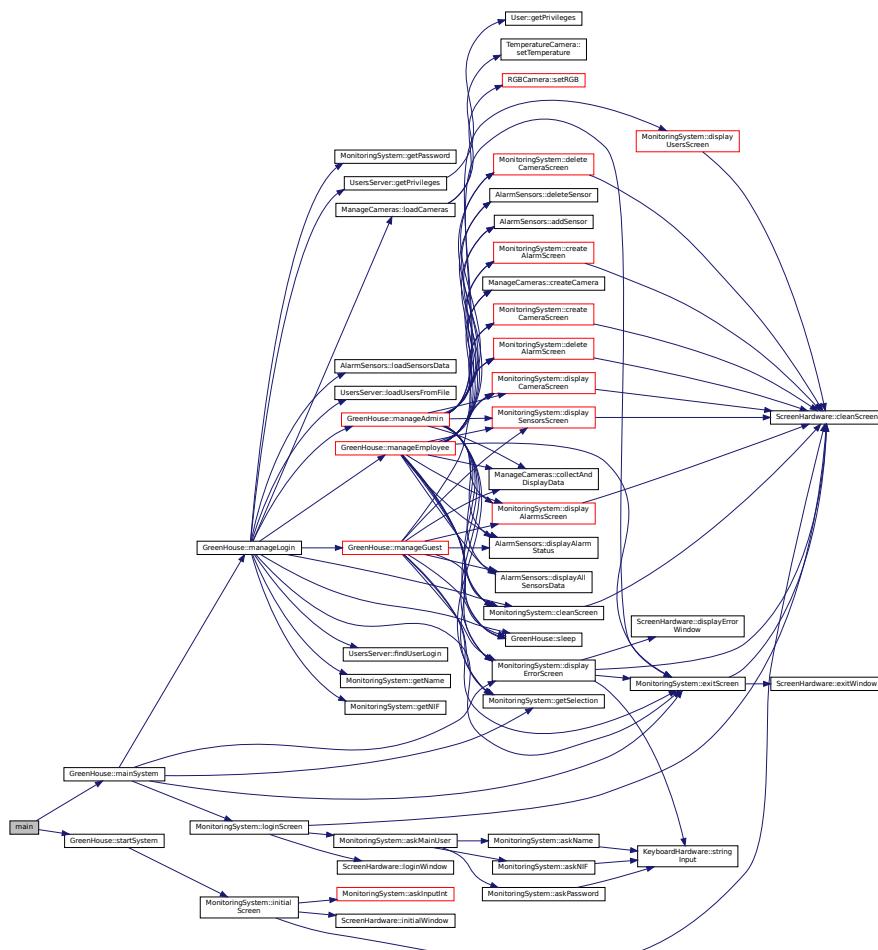
```
int main ( )
```

Definition at line 6 of file main.cpp.

```
6  {
7     GreenHouse greenhouse;
8     greenhouse.startSystem();
9     greenhouse.mainSystem();
10    return 0;
11 }
```

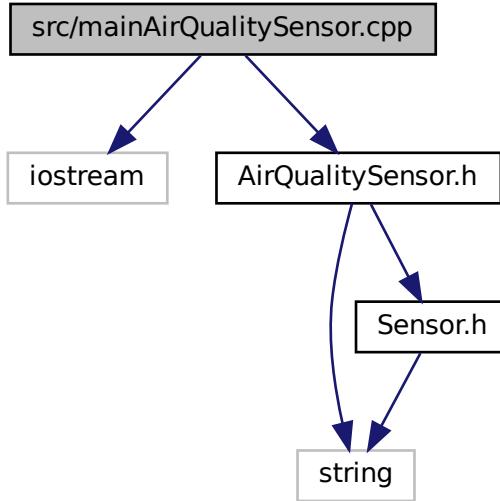
References GreenHouse::mainSystem(), and GreenHouse::startSystem().

Here is the call graph for this function:



## 5.19 src/mainAirQualitySensor.cpp File Reference

```
#include <iostream>
#include "AirQualitySensor.h"
Include dependency graph for mainAirQualitySensor.cpp:
```



## Functions

- int [main \(\)](#)

### 5.19.1 Function Documentation

#### 5.19.1.1 [main\(\)](#)

```
int main ( )
```

Definition at line 6 of file mainAirQualitySensor.cpp.

```

6
7 // Genero un sensor tipo calidad del aire
8 AirQualitySensor airQualitySensor(1, true);
9 // Imprimo la calidad del aire por defecto
10 airQualitySensor.printData();
11 // Cambio el valor de la calidad del aire
12 airQualitySensor.collectData();
13 // Imprimo la nueva calidad del aire
14 airQualitySensor.printData();
15 // Vuelvo a imprimir la calidad del aire
16 cout << "Air Quality: " << airQualitySensor.getData() << endl;
17 cout << "Status: " << airQualitySensor.stringStatus() << endl;
18
19 airQualitySensor.collectData();
```

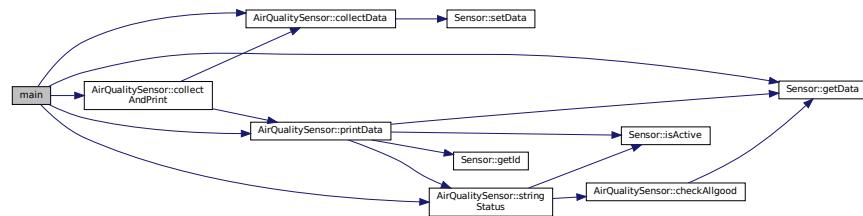
```

20 // Imprimo el sensor de nuevo
21 airQualitySensor.printData();
22
23 // Print collect and print
24 airQualitySensor.collectAndPrint();
25 }

```

References `AirQualitySensor::collectAndPrint()`, `AirQualitySensor::collectData()`, `Sensor::getData()`, `AirQualitySensor::printData()`, and `AirQualitySensor::stringStatus()`.

Here is the call graph for this function:

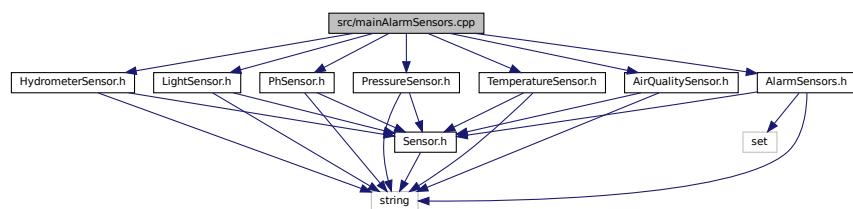


## 5.20 src/mainAlarmSensors.cpp File Reference

```

#include "AirQualitySensor.h"
#include "AlarmSensors.h"
#include "HydrometerSensor.h"
#include "LightSensor.h"
#include "PhSensor.h"
#include "PressureSensor.h"
#include "TemperatureSensor.h"
Include dependency graph for mainAlarmSensors.cpp:

```



## Functions

- int `main ()`

### 5.20.1 Function Documentation

### 5.20.1.1 main()

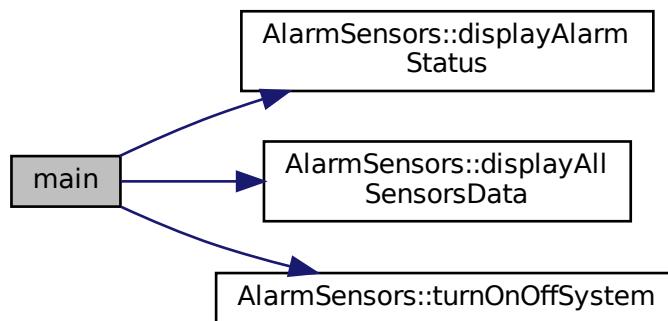
```
int main ( )
```

Definition at line 9 of file mainAlarmSensors.cpp.

```
9      {
10     /*
11     TemperatureSensor temp(1, true);
12     AirQualitySensor air(2, true);
13     HydrometerSensor hyd(3, true);
14     PressureSensor pres(4, true);
15     LightSensor light(5, true);
16     PhSensor ph(6, true);
17     */
18
19     AlarmSensors *alarm = new AlarmSensors(
20         new TemperatureSensor(1, true), new AirQualitySensor(2, true),
21         new HydrometerSensor(3, true), new PressureSensor(4, true),
22         new LightSensor(5, true), new PhSensor(6, true));
23     // Todas las operaciones de la clase AlarmSensors
24     // AlarmSensors alarm(&temp, &air, &hyd, &pres, &light, &ph);
25     alarm->displayAlarmStatus();
26     alarm->displayAllSensorsData();
27     alarm->displayAlarmStatus();
28     alarm->turnOnOffSystem(0);
29     alarm->displayAllSensorsData();
30     alarm->displayAlarmStatus();
31
32     return 0;
33 }
```

References AlarmSensors::displayAlarmStatus(), AlarmSensors::displayAllSensorsData(), and AlarmSensors::turnOnOffSystem().

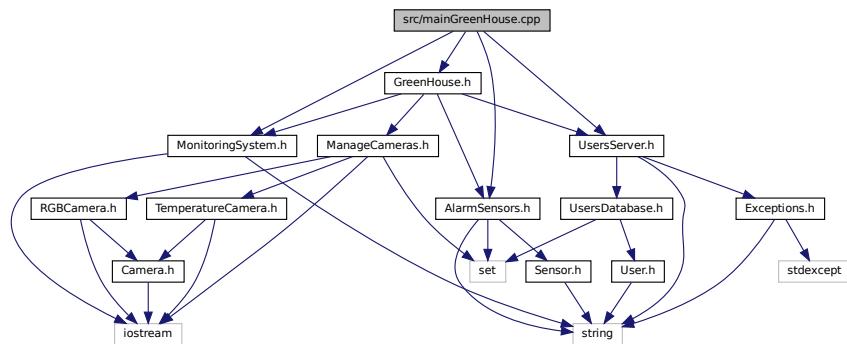
Here is the call graph for this function:



## 5.21 src/mainGreenHouse.cpp File Reference

```
#include "AlarmSensors.h"
#include "GreenHouse.h"
#include "MonitoringSystem.h"
```

```
#include "UsersServer.h"
Include dependency graph for mainGreenHouse.cpp:
```



## Functions

- int `main ()`

### 5.21.1 Function Documentation

#### 5.21.1.1 `main()`

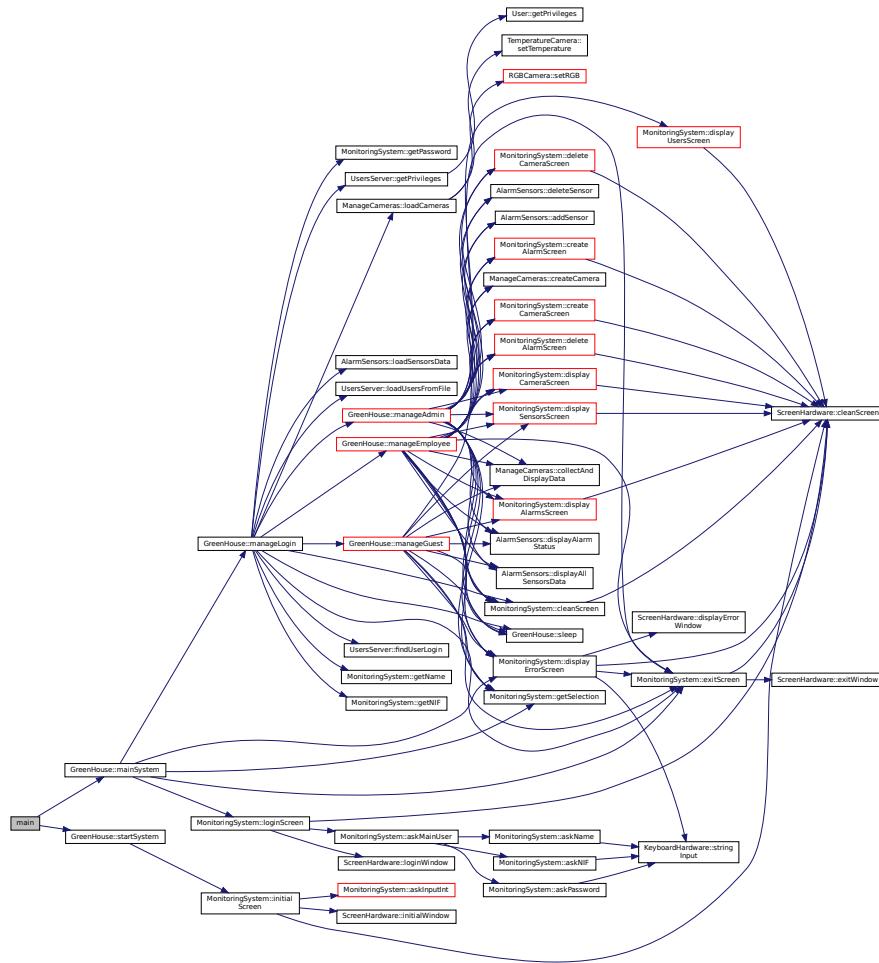
```
int main ( )
```

Definition at line 6 of file `mainGreenHouse.cpp`.

```
6  {
7  GreenHouse gh;
8  gh.startSystem();
9  gh.mainSystem();
10 return 0;
11 }
```

References `GreenHouse::mainSystem()`, and `GreenHouse::startSystem()`.

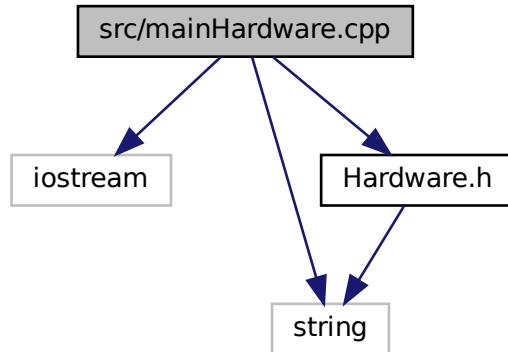
Here is the call graph for this function:



## 5.22 src/mainHardware.cpp File Reference

```
#include <iostream>
#include <string>
#include "Hardware.h"
```

Include dependency graph for mainHardware.cpp:



## Functions

- int [main \(\)](#)

### 5.22.1 Function Documentation

#### 5.22.1.1 [main\(\)](#)

```
int main ( )
```

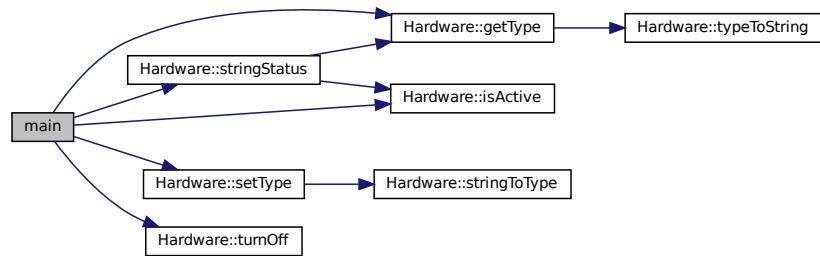
Definition at line 7 of file [mainHardware.cpp](#).

```

7
8 // Aquí tengo que probar los distintos métodos de un objeto Hardware
9 Hardware hardware(true, Hardware::Types_Hardware::SWITCH);
10 // Pruebas get type set type
11 std::cout << hardware.getType() << std::endl;
12 hardware.setType("SCREEN");
13 std::cout << hardware.getType() << std::endl;
14
15 // Vamos a ver si está activado y luego desactivamos
16 if (hardware.isActive()) {
17     std::cout << "Está activo" << std::endl;
18 } else {
19     std::cout << "No está activo" << std::endl;
20 }
21 hardware.turnOff();
22 if (hardware.isActive()) {
23     std::cout << "Está activo" << std::endl;
24 } else {
25     std::cout << "No está activo" << std::endl;
26 }
27 std::cout << hardware.stringStatus() << std::endl;
28 }
```

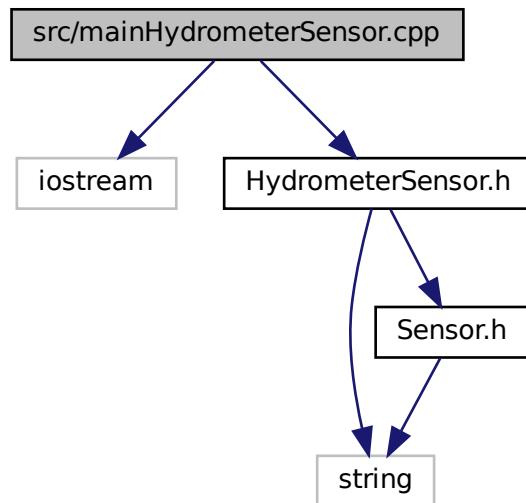
References [Hardware::getType\(\)](#), [Hardware::isActive\(\)](#), [Hardware::setType\(\)](#), [Hardware::stringStatus\(\)](#), and [Hardware::turnOff\(\)](#).

Here is the call graph for this function:



## 5.23 src/mainHydrometerSensor.cpp File Reference

```
#include <iostream>
#include "HydrometerSensor.h"
Include dependency graph for mainHydrometerSensor.cpp:
```



### Functions

- int [main \(\)](#)

#### 5.23.1 Function Documentation

### 5.23.1.1 main()

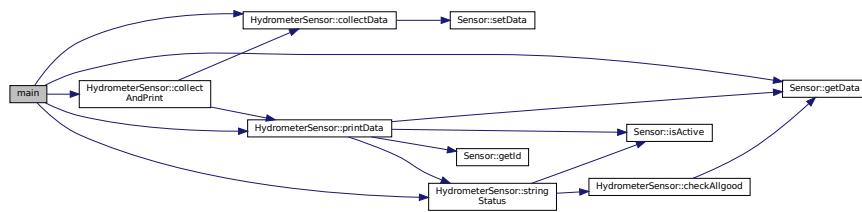
```
int main ( )
```

Definition at line 6 of file mainHydrometerSensor.cpp.

```
6     {
7     // Genero un sensor tipo hidrometro (mide la humedad del aire)
8     HydrometerSensor hydrometer(1, true);
9     // Imprimo la hidrometro (mide la humedad del aire) por defecto
10    hydrometer.printData();
11    // Cambio el valor de la hidrometro (mide la humedad del aire)
12    hydrometer.collectData();
13    // Imprimo la nueva hidrometro (mide la humedad del aire)
14    hydrometer.printData();
15    // Vuelvo a imprimir la hidrometro (mide la humedad del aire)
16    cout << "Air Quality: " << hydrometer.getData() << endl;
17    cout << "Status: " << hydrometer.stringStatus() << endl;
18
19    hydrometer.collectData();
20    // Imprimo el sensor de nuevo
21    hydrometer.printData();
22
23    // Print collect and print
24    hydrometer.collectAndPrint();
25 }
```

References HydrometerSensor::collectAndPrint(), HydrometerSensor::collectData(), Sensor::getData(), HydrometerSensor::printData(), and HydrometerSensor::stringStatus().

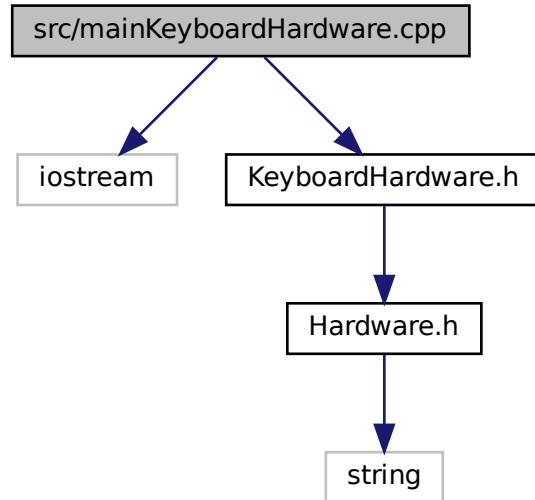
Here is the call graph for this function:



## 5.24 src/mainKeyboardHardware.cpp File Reference

```
#include <iostream>
#include "KeyboardHardware.h"
```

Include dependency graph for mainKeyboardHardware.cpp:



## Functions

- int `main ()`

### 5.24.1 Function Documentation

#### 5.24.1.1 `main()`

```
int main ( )
```

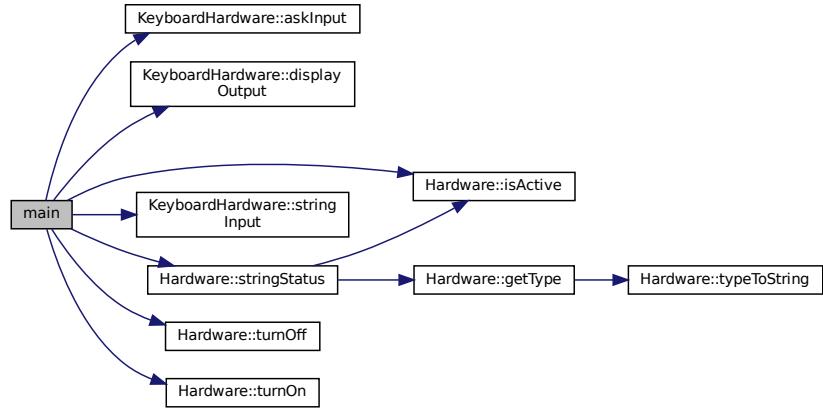
Definition at line 6 of file `mainKeyboardHardware.cpp`.

```

6   {
7     KeyboardHardware keyboard(true);
8     if (keyboard.isActive()) {
9       std::cout << "Active" << std::endl;
10    } else {
11      std::cout << "No Active" << std::endl;
12    }
13    // apagamos
14    keyboard.turnOff();
15    std::cout << keyboard.stringStatus() << std::endl;
16    // encnedemos
17    keyboard.turnOn();
18    std::cout << keyboard.stringStatus() << std::endl;
19
20    // Preguntamos un input y luego mostramos el ultimo input
21    std::cout << keyboard.askInput() << std::endl;
22    std::cout << keyboard.stringInput() << std::endl;
23    keyboard.displayOutput();
24
25    return 0;
26 }
```

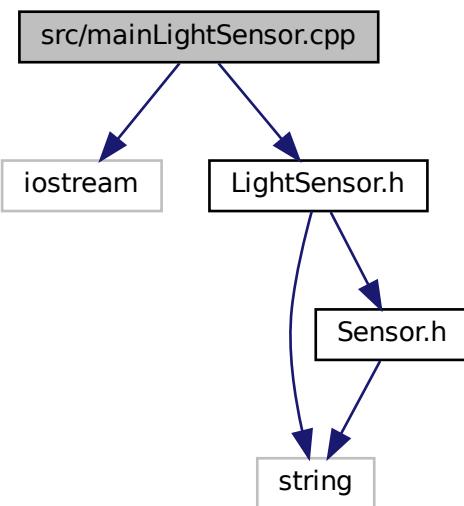
References KeyboardHardware::askInput(), KeyboardHardware::displayOutput(), Hardware::isActive(), KeyboardHardware::stringInput(), Hardware::stringStatus(), Hardware::turnOff(), and Hardware::turnOn().

Here is the call graph for this function:



## 5.25 src/mainLightSensor.cpp File Reference

```
#include <iostream>
#include "LightSensor.h"
Include dependency graph for mainLightSensor.cpp:
```



## Functions

- int `main ()`

## 5.25.1 Function Documentation

### 5.25.1.1 main()

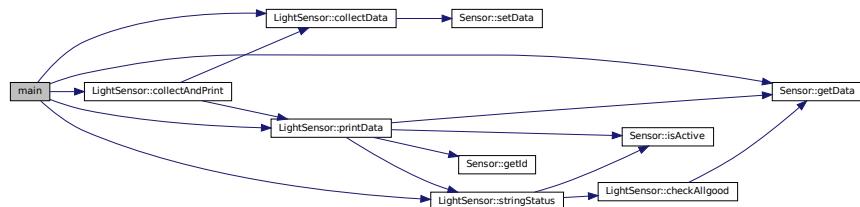
```
int main ( )
```

Definition at line 5 of file mainLightSensor.cpp.

```
5   {
6     LightSensor lightSensor(1, true);
7     // Imprimo la luz por defecto
8     lightSensor.printData();
9     // Cambio el valor de la luz
10    lightSensor.collectData();
11    // Imprimo la nueva luz
12    lightSensor.printData();
13    // Print collect and print
14    std::cout << "Light: " << lightSensor.getData() << std::endl;
15    // Print collect and print
16    std::cout << "Status: " << lightSensor.stringStatus() << std::endl;
17    // Imprimo el sensor de nuevo
18    lightSensor.collectData();
19    // Imprimo el sensor de nuevo
20    lightSensor.printData();
21    // Print collect and print
22    lightSensor.collectAndPrint();
23
24    return 0;
25 }
```

References LightSensor::collectAndPrint(), LightSensor::collectData(), Sensor::getData(), LightSensor::printData(), and LightSensor::stringStatus().

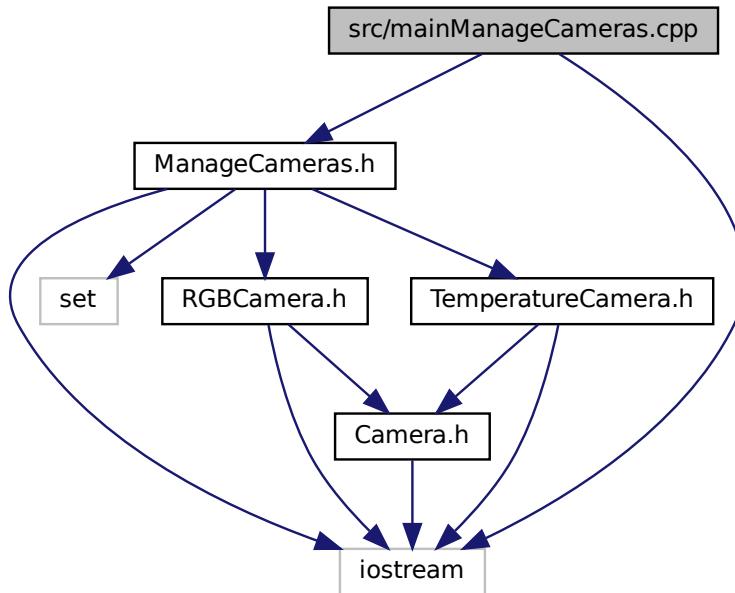
Here is the call graph for this function:



## 5.26 src/mainManageCameras.cpp File Reference

```
#include "ManageCameras.h"
#include <iostream>
```

Include dependency graph for mainManageCameras.cpp:



## Functions

- int `main ()`

### 5.26.1 Function Documentation

#### 5.26.1.1 `main()`

```
int main ( )
```

Definition at line 4 of file `mainManageCameras.cpp`.

```
4   {
5     ManageCameras manageCameras;
6     manageCameras.createCamera(3, "RGB");
7     manageCameras.createCamera(4, "Temperature");
8     manageCameras.displayAllCameras();
9     manageCameras.collectData();
10    manageCameras.displayAllCameras();
11    manageCameras.turnOffCameras();
12    manageCameras.displayAllCameras();
13    manageCameras.removeCamera(3);
14    manageCameras.displayAllCameras();
15    manageCameras.createCamera(4, "RGB");
16    manageCameras.displayAllCameras();
17    manageCameras.createCamera(5, "Temperature");
18    manageCameras.createCamera(5, "Rgb");
19    manageCameras.displayAllCameras();
20    manageCameras.saveCameras();
21    // Limpiar terminal
```

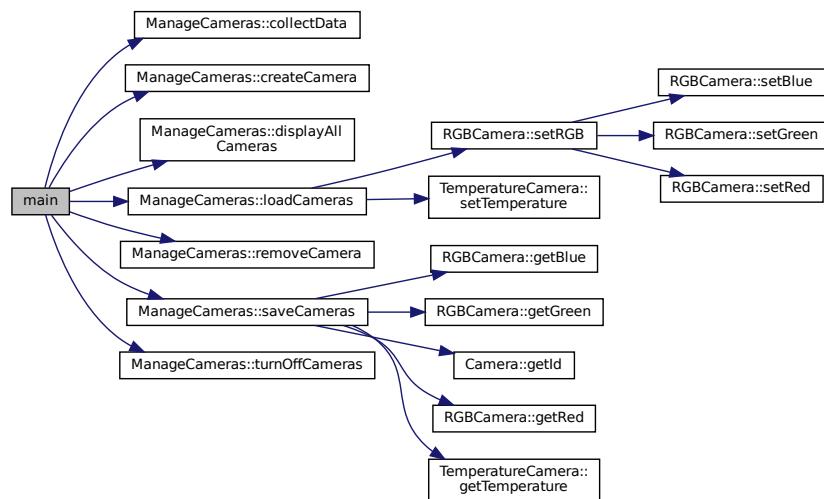
```

22 std::cout << "\033[2J\033[1;1H";
23 ManageCameras manageCameras2;
24 manageCameras2.loadCameras();
25 manageCameras2.displayAllCameras();
26
27 return 0;
28 }

```

References ManageCameras::collectData(), ManageCameras::createCamera(), ManageCameras::displayAllCameras(), ManageCameras::loadCameras(), ManageCameras::removeCamera(), ManageCameras::saveCameras(), and ManageCameras::turnOffCameras().

Here is the call graph for this function:

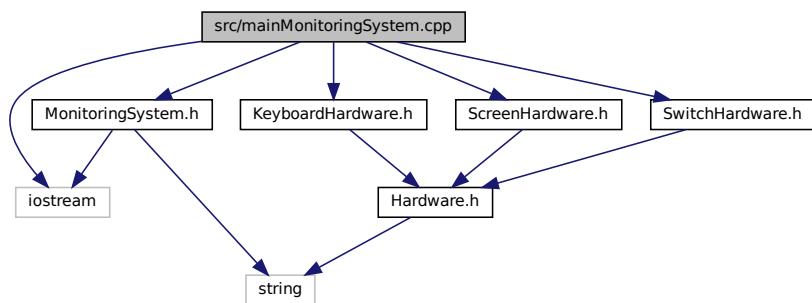


## 5.27 src/mainMonitoringSystem.cpp File Reference

```

#include <iostream>
#include "KeyboardHardware.h"
#include "MonitoringSystem.h"
#include "ScreenHardware.h"
#include "SwitchHardware.h"
Include dependency graph for mainMonitoringSystem.cpp:

```



## Functions

- int `main ()`

### 5.27.1 Function Documentation

#### 5.27.1.1 main()

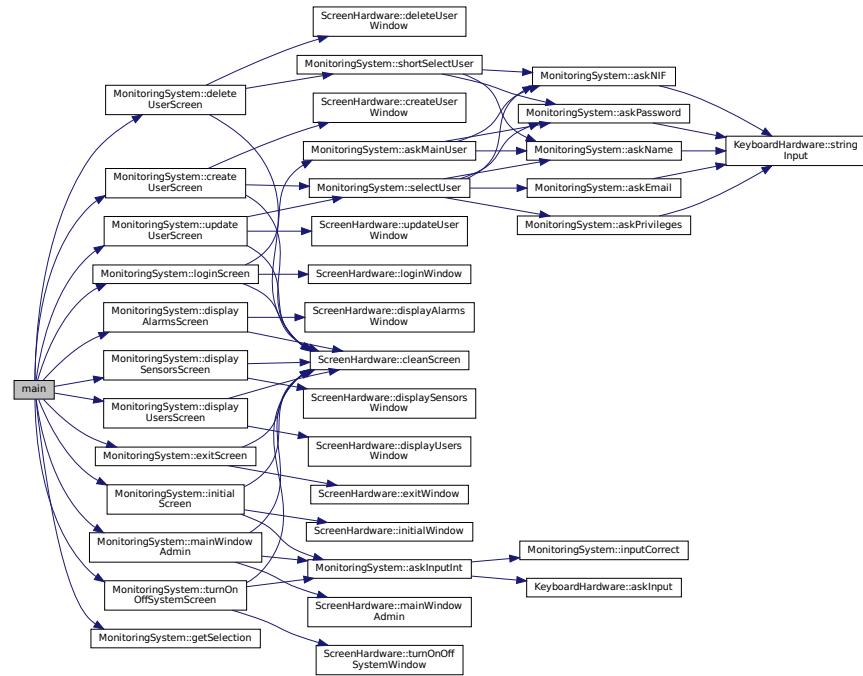
```
int main ( )
```

Definition at line 9 of file mainMonitoringSystem.cpp.

```
9      {
10     // Creo un MonitoringSystem
11     MonitoringSystem *ms =
12         new MonitoringSystem(new ScreenHardware(true), new KeyboardHardware(true),
13                             new SwitchHardware(true));
14     // Llamo a la funcion initialScreen() que muestra el menu inicial
15     ms->initialScreen();
16     if (ms->getSelection() == 1) {
17         ms->loginScreen();
18         // (SUPONEMOS que ha entrado bien el usuario y que es admin)
19         bool exit = false;
20         do {
21             ms->mainWindowAdmin();
22             // Ahora probamos la ventana de employee
23             // ms->mainWindowEmployee();
24             // Ahora probamos la ventana de guest
25             // ms->mainWindowGuest();
26             if (ms->getSelection() == 1) {
27                 ms->createUserScreen();
28             } else if (ms->getSelection() == 2) {
29                 ms->deleteUserScreen();
30             } else if (ms->getSelection() == 3) {
31                 ms->updateUserScreen();
32             } else if (ms->getSelection() == 4) {
33                 ms->displayUsersScreen();
34             } else if (ms->getSelection() == 5) {
35                 ms->displaySensorsScreen();
36             } else if (ms->getSelection() == 6) {
37                 ms->displayAlarmsScreen();
38             } else if (ms->getSelection() == 7) {
39                 ms->turnOnOffSystemScreen();
40             } else if (ms->getSelection() == 8) {
41                 ms->exitScreen();
42                 exit = true;
43             }
44         } while (!exit);
45
46     /*ms->createUserScreen();
47     ms->deleteUserScreen();
48     ms->updateUserScreen();
49     ms->displayUsersScreen();
50     ms->displaySensorsScreen();
51     ms->displayAlarmsScreen();
52     ms->turnOnOffSystemScreen();
53     ms->displayErrorScreen();*/
54     } else {
55         // Llamo a la funcion exitScreen() que muestra el exitWindow
56         ms->exitScreen();
57     }
58
59     return 0;
60 }
```

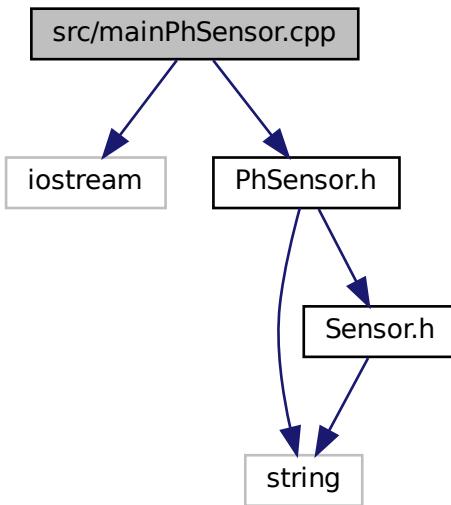
References `MonitoringSystem::createUserScreen()`, `MonitoringSystem::deleteUserScreen()`, `MonitoringSystem::displayAlarmsScreen()`, `MonitoringSystem::displaySensorsScreen()`, `MonitoringSystem::displayUsersScreen()`, `MonitoringSystem::exitScreen()`, `MonitoringSystem::getSelection()`, `MonitoringSystem::initialScreen()`, `MonitoringSystem::loginScreen()`, `MonitoringSystem::mainWindowAdmin()`, `MonitoringSystem::turnOnOffSystemScreen()`, and `MonitoringSystem::updateUserScreen()`.

Here is the call graph for this function:



## 5.28 src/mainPhSensor.cpp File Reference

```
#include <iostream>
#include "PhSensor.h"
Include dependency graph for mainPhSensor.cpp:
```



## Functions

- int main ()

### 5.28.1 Function Documentation

#### 5.28.1.1 main()

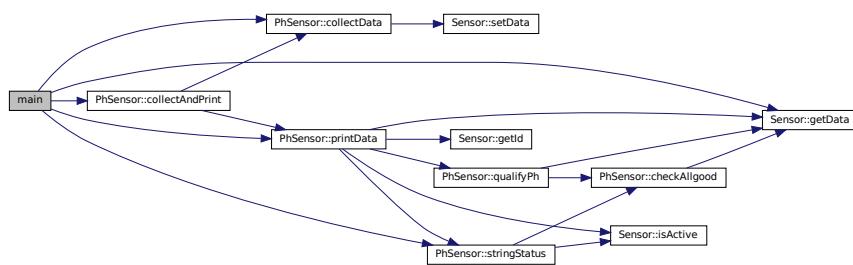
```
int main ( )
```

Definition at line 6 of file mainPhSensor.cpp.

```
6     {
7     PhSensor phsensor(1, true);
8     // Imprimo el ph por defecto
9     phsensor.printData();
10    // Cambio el valor de la luz
11    phsensor.collectData();
12    // Imprimo la nueva luz
13    phsensor.printData();
14    // Print collect and print
15    std::cout << "PH: " << phsensor.getData() << std::endl;
16    // Print collect and print
17    std::cout << "Status: " << phsensor.stringStatus() << std::endl;
18    // Imprimo el sensor de nuevo
19    phsensor.collectData();
20    // Imprimo el sensor de nuevo
21    phsensor.printData();
22    // Print collect and print
23    phsensor.collectAndPrint();
24 }
```

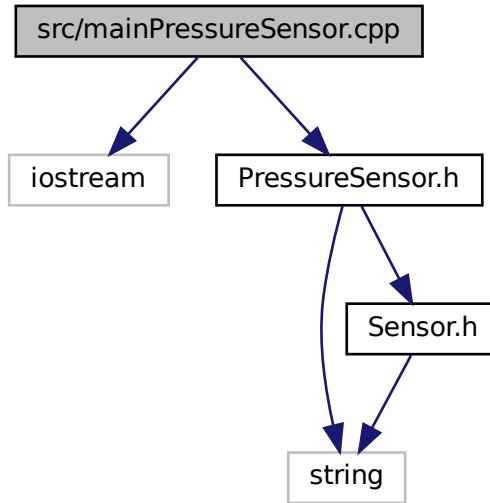
References PhSensor::collectAndPrint(), PhSensor::collectData(), Sensor::getData(), PhSensor::printData(), and PhSensor::stringStatus().

Here is the call graph for this function:



## 5.29 src/mainPressureSensor.cpp File Reference

```
#include <iostream>
#include "PressureSensor.h"
Include dependency graph for mainPressureSensor.cpp:
```



### Functions

- int `main ()`

#### 5.29.1 Function Documentation

##### 5.29.1.1 `main()`

```
int main ( )
```

Definition at line 6 of file mainPressureSensor.cpp.

```

6  {
7   PressureSensor pressureSensor(1, true);
8   // Imprimo la presion por defecto
9   pressureSensor.printData();
10  // Cambio el valor de la presion
11  pressureSensor.collectData();
12  // Imprimo la nueva presion
13  pressureSensor.printData();
14  // Vuelvo a imprimir la presion
15  cout << "Pressure: " << pressureSensor.getData() << endl;
16  cout << "Status: " << pressureSensor.stringStatus() << endl;
17  pressureSensor.collectData();
18  // Imprimo el sensor de nuevo
19  pressureSensor.printData();
```

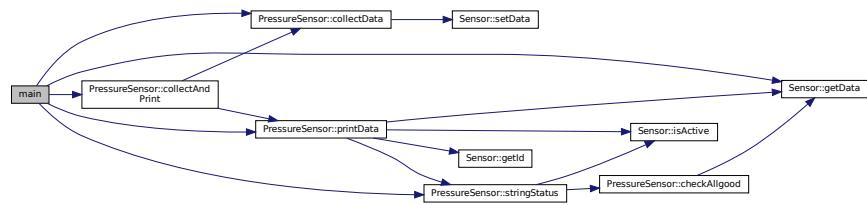
```

20 // Print collect and print
21 pressureSensor.collectAndPrint();
22 }

```

References PressureSensor::collectAndPrint(), PressureSensor::collectData(), Sensor::getData(), PressureSensor::printData(), and PressureSensor::stringStatus().

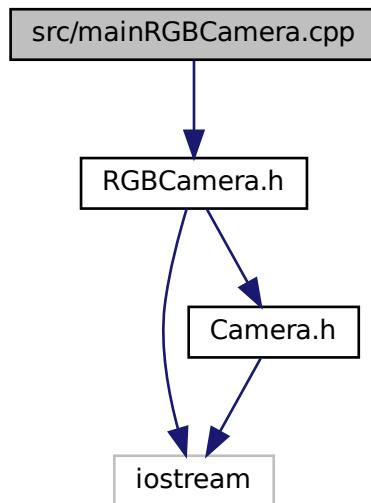
Here is the call graph for this function:



## 5.30 src/mainRGBCamera.cpp File Reference

```
#include "RGBCamera.h"
```

Include dependency graph for mainRGBCamera.cpp:



## Functions

- int `main ()`

### 5.30.1 Function Documentation

### 5.30.1.1 main()

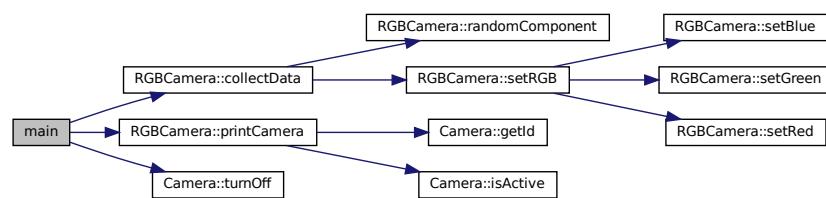
```
int main ( )
```

Definition at line 3 of file mainRGBCamera.cpp.

```
3   {
4     RGBCamera cameral(1);
5     cameral.printCamera();
6     cameral.collectData();
7     cameral.printCamera();
8     cameral.turnOff();
9     cameral.printCamera();
10    return 0;
11 }
```

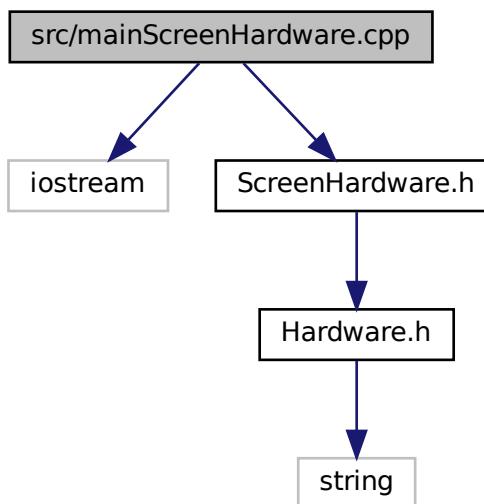
References RGBCamera::collectData(), RGBCamera::printCamera(), and Camera::turnOff().

Here is the call graph for this function:



## 5.31 src/mainScreenHardware.cpp File Reference

```
#include <iostream>
#include "ScreenHardware.h"
Include dependency graph for mainScreenHardware.cpp:
```



## Functions

- int `main ()`

### 5.31.1 Function Documentation

#### 5.31.1.1 main()

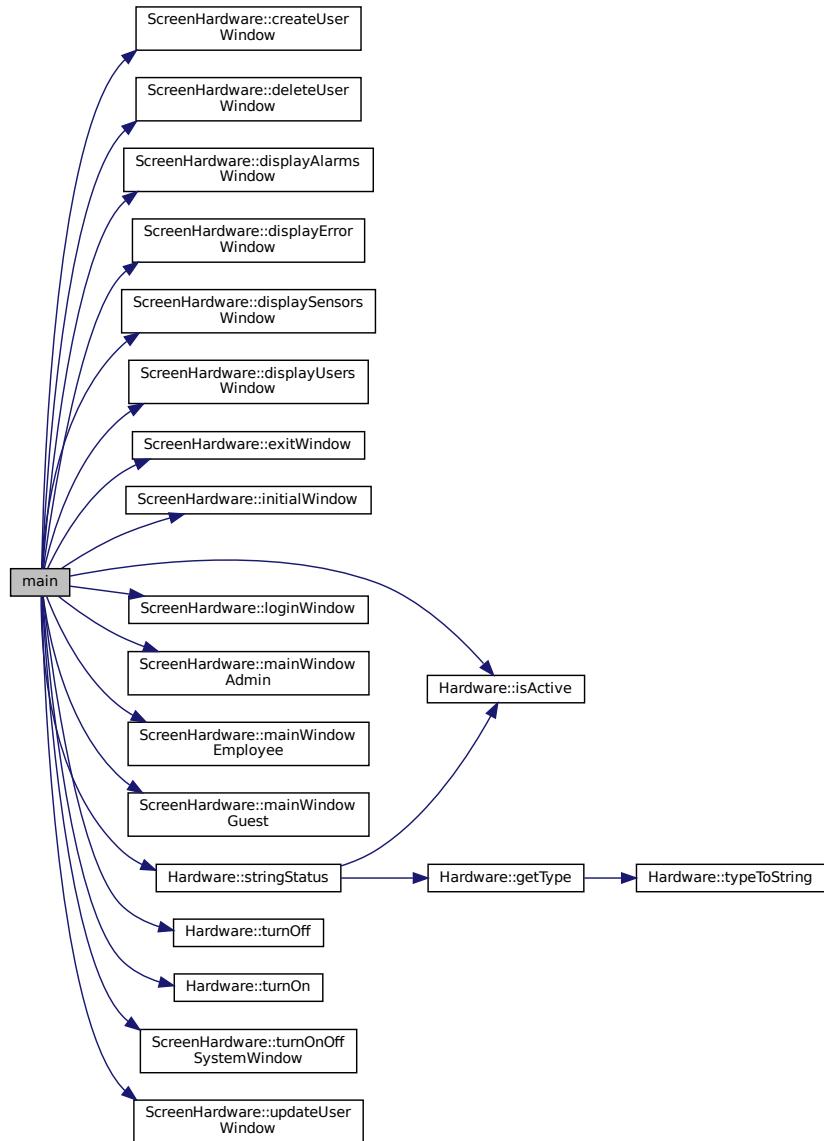
```
int main ( )
```

Definition at line 6 of file mainScreenHardware.cpp.

```
6      {
7      ScreenHardware screenHardware(true);
8      if (screenHardware.isActive()) {
9          std::cout << "Active" << std::endl;
10     } else {
11         std::cout << "No Active" << std::endl;
12     }
13     // apagamos
14     screenHardware.turnOff();
15     std::cout << screenHardware.stringStatus() << std::endl;
16     // encnedemos
17     screenHardware.turnOn();
18     std::cout << screenHardware.stringStatus() << std::endl;
19     // Ahora mostramos cada tipo de pantalla para comprobar que funciona
20     screenHardware.initialWindow();
21     screenHardware.loginWindow();
22     screenHardware.mainWindowAdmin();
23     screenHardware.mainWindowEmployee();
24     screenHardware.mainWindowGuest();
25     screenHardware.createUserWindow();
26     screenHardware.deleteUserWindow();
27     screenHardware.updateUserWindow();
28     screenHardware.displayUsersWindow();
29     screenHardware.displaySensorsWindow();
30     screenHardware.displayAlarmsWindow();
31     screenHardware.turnOnOffSystemWindow();
32     screenHardware.displayErrorWindow();
33     screenHardware.exitWindow();
34 }
```

References `ScreenHardware::createUserWindow()`, `ScreenHardware::deleteUserWindow()`, `ScreenHardware::displayAlarmsWindow()`, `ScreenHardware::displayErrorWindow()`, `ScreenHardware::displaySensorsWindow()`, `ScreenHardware::displayUsersWindow()`, `ScreenHardware::exitWindow()`, `ScreenHardware::initialWindow()`, `Hardware::isActive()`, `ScreenHardware::loginWindow()`, `ScreenHardware::mainWindowAdmin()`, `ScreenHardware::mainWindowEmployee()`, `ScreenHardware::mainWindowGuest()`, `Hardware::stringStatus()`, `Hardware::turnOff()`, `Hardware::turnOn()`, `ScreenHardware::turnOnOffSystemWindow()`, and `ScreenHardware::updateUserWindow()`.

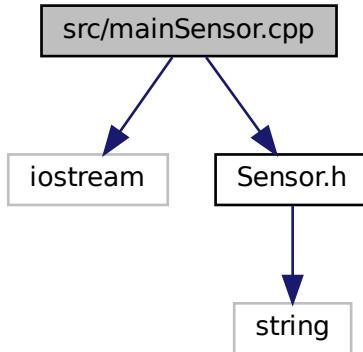
Here is the call graph for this function:



## 5.32 src/mainSensor.cpp File Reference

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

Include dependency graph for mainSensor.cpp:



## Functions

- int `main ()`

### 5.32.1 Function Documentation

#### 5.32.1.1 main()

```
int main ( )
```

Definition at line 6 of file mainSensor.cpp.

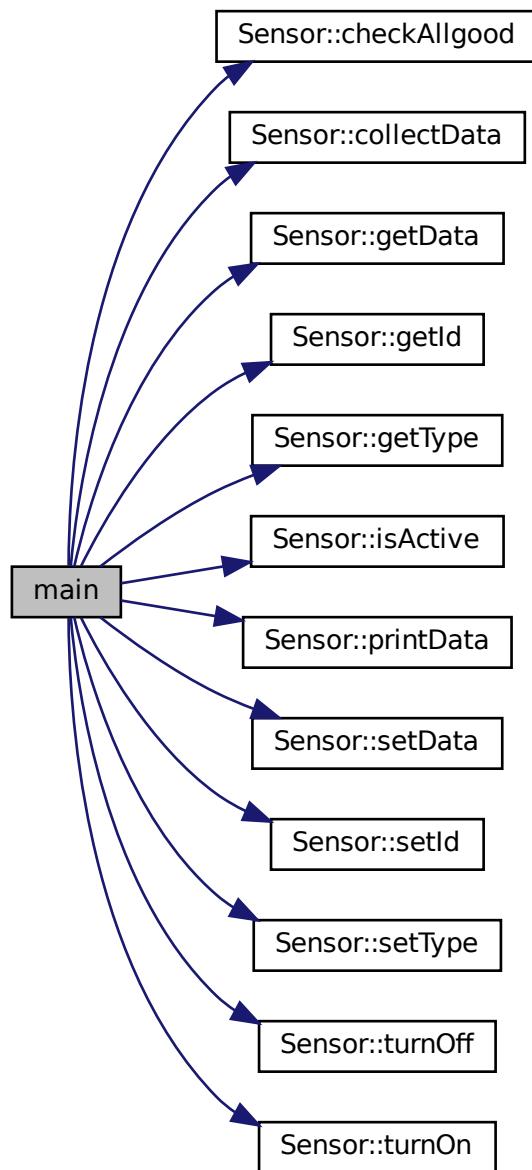
```

6
7 // Creamos un sensor de cada tipo
8 Sensor tempSensor(1, Sensor::Types::TEMPERATURE, true);
9 Sensor airSensor(2, Sensor::Types::AIR_QUALITY, true);
10 Sensor pressureSensor(5, Sensor::Types::PRESSURE, true);
11 Sensor hSensor(6, Sensor::Types::HYDROMETER, true);
12 Sensor lightSensor(3, Sensor::Types::LIGHT_SENSOR, true);
13 Sensor phSensor(4, Sensor::Types::PH_SENSOR, true);
14
15 // Ahora jugamos con los datos de los sensores
16 tempSensor.collectData();
17 cout << "Type: " << tempSensor.getType() << endl;
18 // Ahora apagamos y encendemos
19 tempSensor.turnOff();
20 if (tempSensor.isActive()) {
21     cout << "Sensor is active" << endl;
22 } else {
23     cout << "Sensor is inactive" << std::endl;
24 }
25 tempSensor.turnOn();
26 if (tempSensor.isActive()) {
27     cout << "Sensor is active" << endl;
28 } else {
29     cout << "Sensor is inactive" << std::endl;
30 }
31 // Asignamos un valor a la variable data_
32 tempSensor.setData(25.5);
33 cout << "Data: " << tempSensor.getData() << endl;
```

```
34 // Cambiamos el id
35 tempSensor.setId(10);
36 cout << "ID: " << tempSensor.getId() << endl;
37 // Cambiamos el tipo
38 tempSensor.setType("AIR_QUALITY");
39 cout << "Type: " << tempSensor.getType() << endl;
40
41 // Print data
42 tempSensor.printData();
43 if (tempSensor.checkAllgood()) {
44     cout << "All good!" << endl;
45 } else {
46     cout << "Not all good" << endl;
47 }
48
49 return 0;
50 }
```

References Sensor::checkAllgood(), Sensor::collectData(), Sensor::getData(), Sensor::getId(), Sensor::getType(), Sensor::isActive(), Sensor::printData(), Sensor::setData(), Sensor::setId(), Sensor::setType(), Sensor::turnOff(), and Sensor::turnOn().

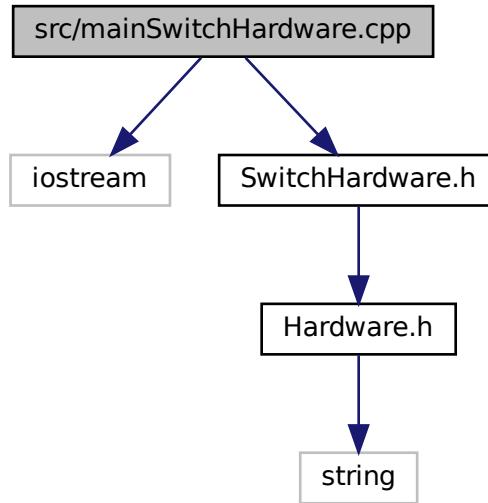
Here is the call graph for this function:



## 5.33 src/mainSwitchHardware.cpp File Reference

```
#include <iostream>
#include "SwitchHardware.h"
```

Include dependency graph for mainSwitchHardware.cpp:



## Functions

- int `main ()`

### 5.33.1 Function Documentation

#### 5.33.1.1 `main()`

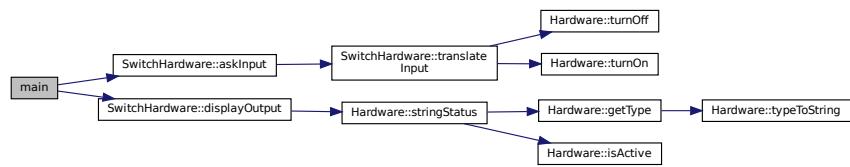
```
int main ( )
```

Definition at line 6 of file `mainSwitchHardware.cpp`.

```
6  {
7  // Generamos un switch activado, luego preguntamos un input, displayamos el
8  // output, y luego repetimos
9  SwitchHardware sw(true);
10 sw.displayOutput();
11 sw.askInput();
12 sw.displayOutput();
13 sw.askInput();
14 sw.displayOutput();
15 }
```

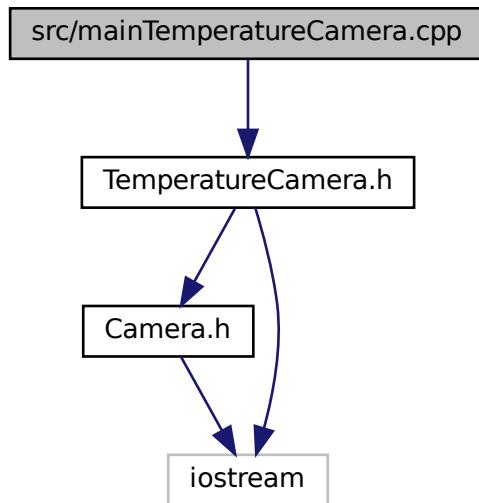
References `SwitchHardware::askInput()`, and `SwitchHardware::displayOutput()`.

Here is the call graph for this function:



## 5.34 src/mainTemperatureCamera.cpp File Reference

```
#include "TemperatureCamera.h"  
Include dependency graph for mainTemperatureCamera.cpp:
```



## Functions

- int `main ()`

### 5.34.1 Function Documentation

### 5.34.1.1 main()

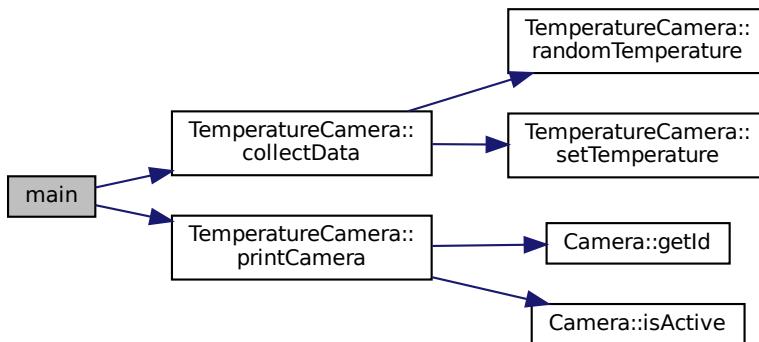
```
int main ( )
```

Definition at line 3 of file mainTemperatureCamera.cpp.

```
3     {
4     TemperatureCamera cameral(1);
5     cameral.printCamera();
6     cameral.collectData();
7     cameral.printCamera();
8     return 0;
9 }
```

References TemperatureCamera::collectData(), and TemperatureCamera::printCamera().

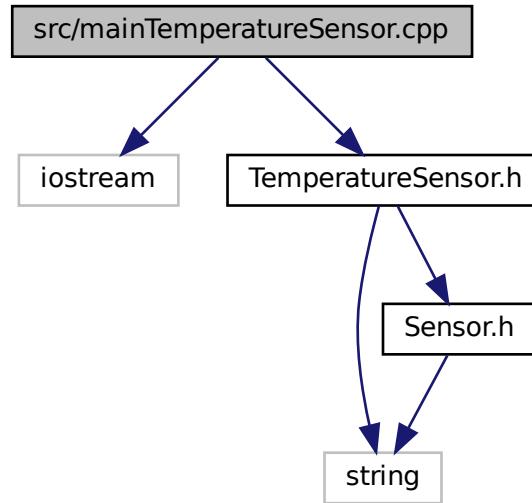
Here is the call graph for this function:



## 5.35 src/mainTemperatureSensor.cpp File Reference

```
#include <iostream>
#include "TemperatureSensor.h"
```

Include dependency graph for mainTemperatureSensor.cpp:



## Functions

- int `main ()`

### 5.35.1 Function Documentation

#### 5.35.1.1 `main()`

```
int main ( )
```

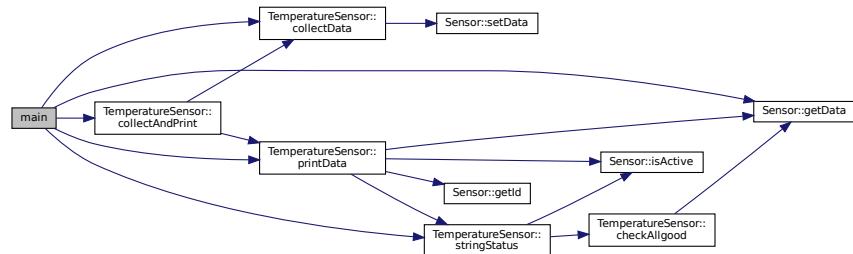
Definition at line 6 of file mainTemperatureSensor.cpp.

```

6   {
7   // Genero un sensor tipo temperatura
8   TemperatureSensor tempSensor(1, true);
9   // Imprimo la temperatura por defecto
10  tempSensor.printData();
11  // Cambio el valor de la temperatura
12  tempSensor.collectData();
13  // Imprimo la nueva temperatura
14  tempSensor.printData();
15  // Vuelvo a imprimir la temperatura
16  cout << "Temperatura: " << tempSensor.getData() << endl;
17  cout << "Status: " << tempSensor.stringStatus() << endl;
18  tempSensor.collectData();
19  // Imprimo el sensor de nuevo
20  tempSensor.printData();
21
22  // Print collect and print
23  tempSensor.collectAndPrint();
24 }
```

References TemperatureSensor::collectAndPrint(), TemperatureSensor::collectData(), Sensor::getData(), TemperatureSensor::printData(), and TemperatureSensor::stringStatus().

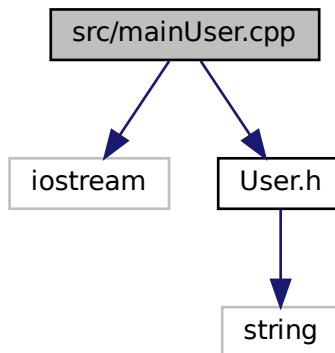
Here is the call graph for this function:



## 5.36 src/mainUser.cpp File Reference

```
#include <iostream>
#include "User.h"
```

Include dependency graph for mainUser.cpp:



## Functions

- int `main ()`

### 5.36.1 Function Documentation

### 5.36.1.1 main()

```
int main ( )
```

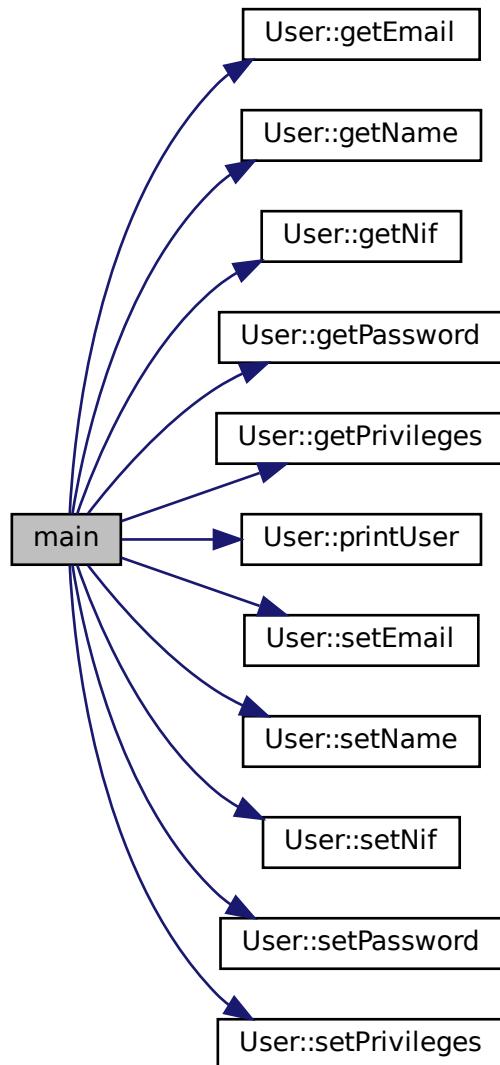
Definition at line 6 of file mainUser.cpp.

```

6      {
7 // Create a user with no name, NIF, password, privileges, and email
8 // Values are assigned to the attributes using setters
9 User user1;
10 user1.setName("Adrian");
11 user1.setNif("234453Y");
12 user1.setPassword("ORANGE_JUICE");
13 user1.setEmail("correoDeAdrian@potatoe.com");
14 user1.setPrivileges("admin");
15 // Print each attribute separately using getters
16 cout << "Name: " << user1.getName() << endl;
17 cout << "NIF: " << user1.getNif() << endl;
18 cout << "Password: " << user1.getPassword() << endl;
19 cout << "Privileges: " << user1.getPrivileges() << endl;
20 cout << "Email: " << user1.getEmail() << endl;
21
22 User user2("Lena", "LJ809K5ES43", "Bagumm.87;", "guest",
23             "liselese.ratte@aol.com");
24 user2.printUser();
25
26 // Compare users
27 // If user 1 is greater than user 2 (privileges)
28 if (user1.operator>(user2)) {
29     cout << "The user " << user1.getName() << " has higher rank than "
30         << user2.getName() << endl;
31 } else {
32     cout << "User " << user1.getName() << " has lower rank than "
33         << user2.getName() << endl;
34 }
35
36 if (user1.operator<(user2)) {
37     cout << "The user " << user1.getName() << " has lower rank than "
38         << user2.getName() << endl;
39 } else {
40     cout << "User " << user1.getName() << " has higher rank than "
41         << user2.getName() << endl;
42 }
43
44 // If user 1 is equal to user 2 (NIF)
45 if (user1.operator==(user2)) {
46     cout << "User " << user1.getName() << " is equal to user "
47         << user2.getName() << endl;
48 } else {
49     cout << "User " << user1.getName() << " is NOT equal to user "
50         << user2.getName() << endl;
51 }
52 // If user 1 is equal to user 1 (name, NIF, and password)
53 if (user1.operator==(user1)) {
54     cout << "User " << user1.getName() << " is equal to user "
55         << user1.getName() << endl;
56 } else {
57     cout << "User " << user1.getName() << " is NOT equal to user "
58         << user1.getName() << endl;
59 }
60
61 // Print a user using std::ostream
62 User user3("Carlos", "010303403L", "1234_password", "employee",
63             "carlos_sainz@gmail.com");
64 cout << "PRINT USER WITH std::ostream" << endl;
65 cout << user3 << endl;
66
67 // Create a user using std::istream
68 User user4;
69 std::cout << "CREATE USER WITH std::istream" << endl;
70 std::cout << "NAME, NIF, PASSWORD, PRIVILEGES, EMAIL" << endl;
71 cin >> user4;
72 user4.printUser();
73 cout << "PRINT USER WITH std::ostream" << endl;
74 cout << user4 << endl;
75
76 return 0;
77 }
```

References User::getEmail(), User::getName(), User::getNif(), User::getPassword(), User::getPrivileges(), User::printUser(), User::setEmail(), User::setName(), User::setNif(), User::setPassword(), and User::setPrivileges().

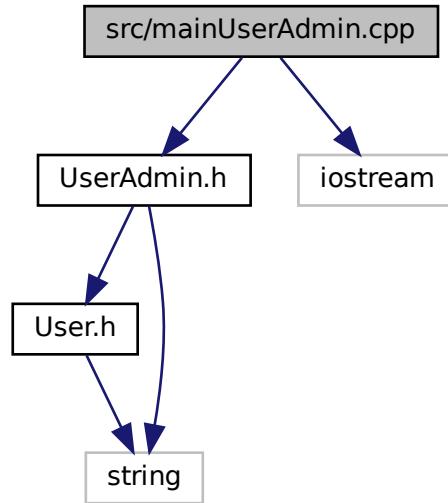
Here is the call graph for this function:



### 5.37 src/mainUserAdmin.cpp File Reference

```
#include "UserAdmin.h"
#include <iostream>
```

Include dependency graph for mainUserAdmin.cpp:



## Functions

- int `main ()`

### 5.37.1 Function Documentation

#### 5.37.1.1 `main()`

```
int main ( )
```

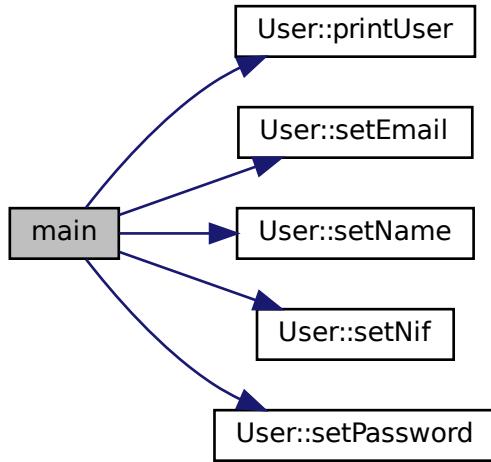
Definition at line 6 of file mainUserAdmin.cpp.

```

6   {
7     // Create a user with no name, NIF, password, privileges, and email
8     // Values are assigned to the attributes using setters
9     UserAdmin user1;
10    user1.setName("Adrian");
11    user1.setNif("234453Y");
12    user1.setPassword("ORANGE_JUICE");
13    user1.setEmail("adrian.adyra@gmail.com");
14    // Imprimir el usuario
15    user1.printUser();
16
17    UserAdmin user2("Lena", "LJ809K5ES43", "Bagumm.87i",
18                  "liselese.ratte@aol.com");
19    user2.printUser();
20 }
```

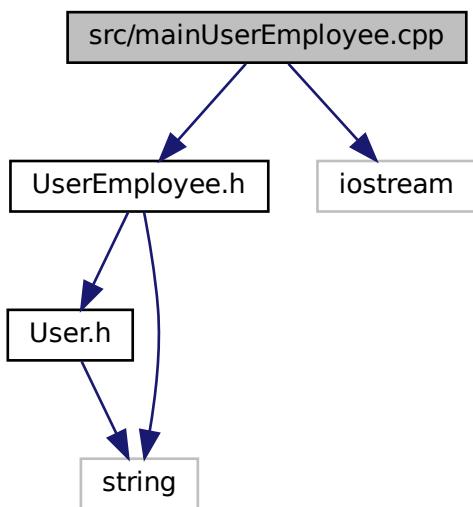
References `User::printUser()`, `User::setEmail()`, `User::setName()`, `User::setNif()`, and `User::setPassword()`.

Here is the call graph for this function:



### 5.38 src/mainUserEmployee.cpp File Reference

```
#include "UserEmployee.h"
#include <iostream>
Include dependency graph for mainUserEmployee.cpp:
```



## Functions

- int `main ()`

### 5.38.1 Function Documentation

#### 5.38.1.1 `main()`

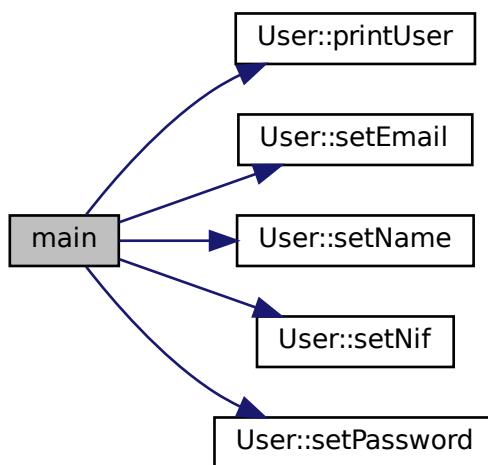
```
int main ( )
```

Definition at line 6 of file mainUserEmployee.cpp.

```
6     {
7     UserEmployee user1("Adrian", "234453Y", "ORANGE_JUICE", "correo");
8     user1.printUser();
9
10    UserEmployee user2;
11    user2.setName("Lena");
12    user2.setNif("LJ809K5ES43");
13    user2.setPassword("Bagumm.87 i");
14    user2.setEmail("correo2");
15    user2.printUser();
16
17    return 0;
18 }
```

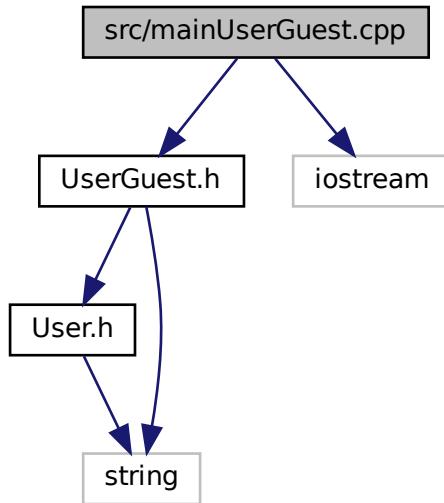
References `User::printUser()`, `User::setEmail()`, `User::setName()`, `User::setNif()`, and `User::setPassword()`.

Here is the call graph for this function:



## 5.39 src/mainUserGuest.cpp File Reference

```
#include "UserGuest.h"
#include <iostream>
Include dependency graph for mainUserGuest.cpp:
```



### Functions

- int `main ()`

#### 5.39.1 Function Documentation

##### 5.39.1.1 `main()`

```
int main ( )
```

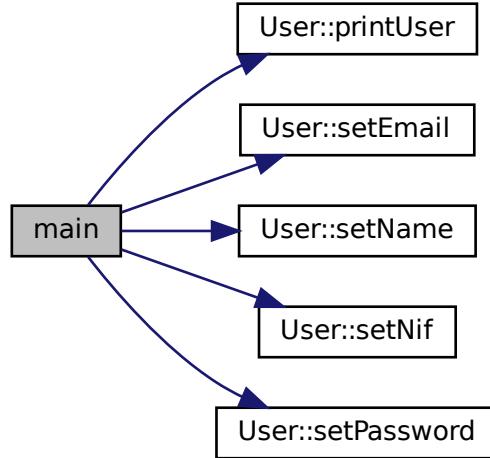
Definition at line 6 of file mainUserGuest.cpp.

```

6   {
7     UserGuest user1("Adrian", "234453Y", "ORANGE_JUICE", "correo");
8     user1.printUser();
9
10    UserGuest user2;
11    user2.setName("Lena");
12    user2.setNif("LJ809K5ES43");
13    user2.setPassword("Bagumm.87i");
14    user2.setEmail("correo2");
15    user2.printUser();
16
17    return 0;
18 }
```

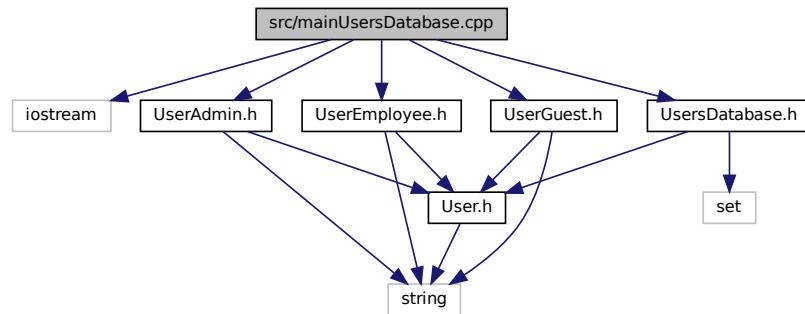
References User::printUser(), User::setEmail(), User::setName(), User::setNif(), and User::setPassword().

Here is the call graph for this function:



## 5.40 src/mainUsersDatabase.cpp File Reference

```
#include <iostream>
#include "UserAdmin.h"
#include "UserEmployee.h"
#include "UserGuest.h"
#include "UsersDatabase.h"
Include dependency graph for mainUsersDatabase.cpp:
```



## Functions

- int [main \(\)](#)

## 5.40.1 Function Documentation

### 5.40.1.1 main()

```
int main ( )
```

Definition at line 8 of file mainUsersDatabase.cpp.

```
8      {
9      // Crear una instancia de UsersDatabase
10     UsersDatabase usersDatabase;
11
12    // Agregar un alumno de cada tipo
13    usersDatabase.addUser(
14        new UserAdmin("Lena", "LJ809K5ES43", "Bagumm.87i", "correoDeAdmin"));
15    usersDatabase.addUser(
16        new UserEmployee("PEPE", "NIFPEPE", "hhhheh.rrrrr", "correoDeEmpleado"));
17    usersDatabase.addUser(new UserGuest(
18        "Juan", "sls1slMINIFG", "sssdqdqadqwdrrwwrrr", "correoDeInvitado"));
19    usersDatabase.addUser(new UserGuest(
20        "Juan", "sls1slMINIFG", "sssdqdqadqwdrrwwrrr", "correoDeInvitado"));
21    usersDatabase.addUser(new UserGuest(
22        "Sujeto -1", "-1", "sssdqdqadqwdrrwwrrr", "correoDeInvitado"));
23    usersDatabase.addUser(new UserGuest(
24        "Sujeto 1", "1", "sssdqdqadqwdrrwwrrr", "correoDeInvitado"));
25    usersDatabase.addUser(new UserGuest(
26        "Sujeto 2", "2", "sssdqdqadqwdrrwwrrr", "correoDeInvitado"));
27    usersDatabase.addUser(
28        new UserGuest("Sujeto 0", "0", "sssdqdqadqwdrrwwrrr", "correo0"));
29
30    // Imprimir los usuarios
31    usersDatabase.printUsers();
32
33    // Obtener una copia de los usuarios
34    UsersDatabase usersDatabaseCopy;
35    std::set<const User *, UserPtrComparator> usersCopy =
36        usersDatabase.getUsers();
37    usersDatabaseCopy.setUsers(usersCopy);
38    std::cout
39        << "\n\n-----\n"
40        << std::endl;
41
42    usersDatabaseCopy.printUsers();
43
44    // Intentamos encontrar un usuario
45    User *user = usersDatabaseCopy.findUser(
46        UserAdmin("Lena", "LJ809K5ES43", "Bagumm.87i", "correoDeAdmin"));
47    if (user) {
48        std::cout << "User found" << std::endl;
49    } else {
50        std::cout << "User not found" << std::endl;
51    }
52
53    // Intentamos encontrar un usuario por nombre
54    user = usersDatabaseCopy.findUserByName("d");
55    if (user) {
56        std::cout << "User found" << std::endl;
57    } else {
58        std::cout << "User not found" << std::endl;
59    }
60
61    // Intentamos encontrar un usuario por NIF
62    user = usersDatabaseCopy.findUserByNif("sls1slMINIFG");
63    if (user) {
64        std::cout << "User found" << std::endl;
65    } else {
66        std::cout << "User not found" << std::endl;
67    }
68
69    // Intentamos encontrar un usuario por correo electrónico
70    user = usersDatabaseCopy.findUserByEmail("correoDeInvitado");
71    if (user) {
72        std::cout << "User found" << std::endl;
73    } else {
74        std::cout << "User not found" << std::endl;
75    }
76
77    // Borrar un usuario
78    usersDatabase.deleteUser(
```

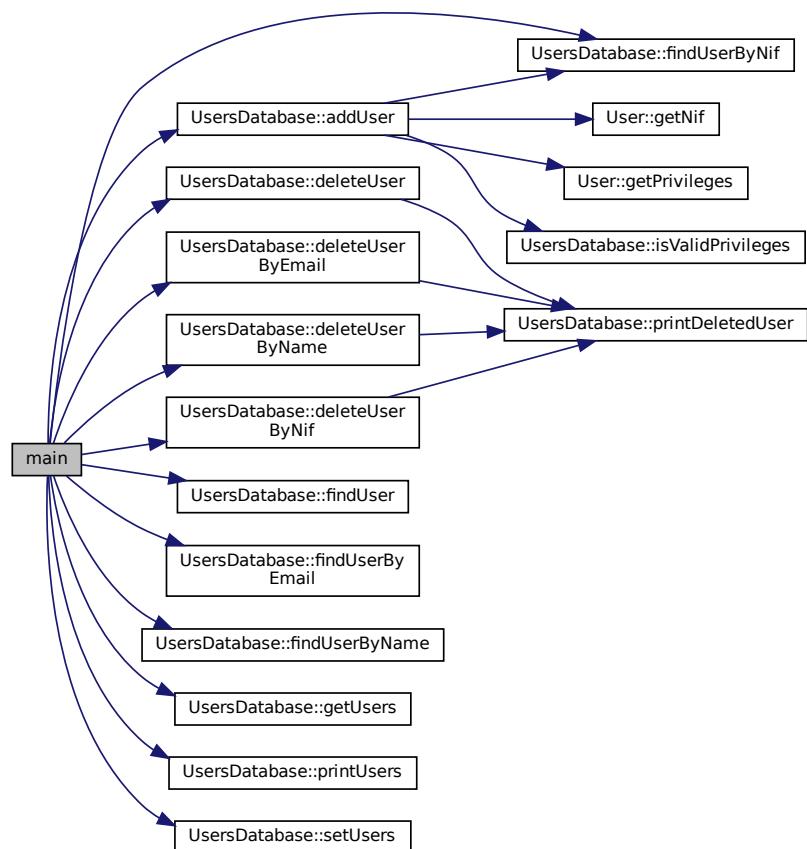
```

79     UserAdmin("Lena", "LJ809K5ES43", "Bagumm.87;", "correoDeAdmin"));
80
81 // Borrar por nombre
82 usersDatabase.deleteUserByName("PEPE");
83
84 // Borrar por NIF
85 usersDatabase.deleteUserByNif("2");
86 usersDatabase.deleteUserByNif("2");
87
88 // Borrar por correo electrónico
89 usersDatabase.deleteUserByEmail("correo0");
90
91 std::cout
92     << "\n\n-----\n\n"
93     << std::endl;
94
95 // Imprimir los usuarios
96 usersDatabase.printUsers();
97
98 return 0;
99 }

```

References UsersDatabase::addUser(), UsersDatabase::deleteUser(), UsersDatabase::deleteUserByEmail(), UsersDatabase::deleteUserByName(), UsersDatabase::deleteUserByNif(), UsersDatabase::findUser(), UsersDatabase::findUserByEmail(), UsersDatabase::findUserByName(), UsersDatabase::findUserByNif(), UsersDatabase::getUsers(), UsersDatabase::printUsers(), and UsersDatabase::setUsers().

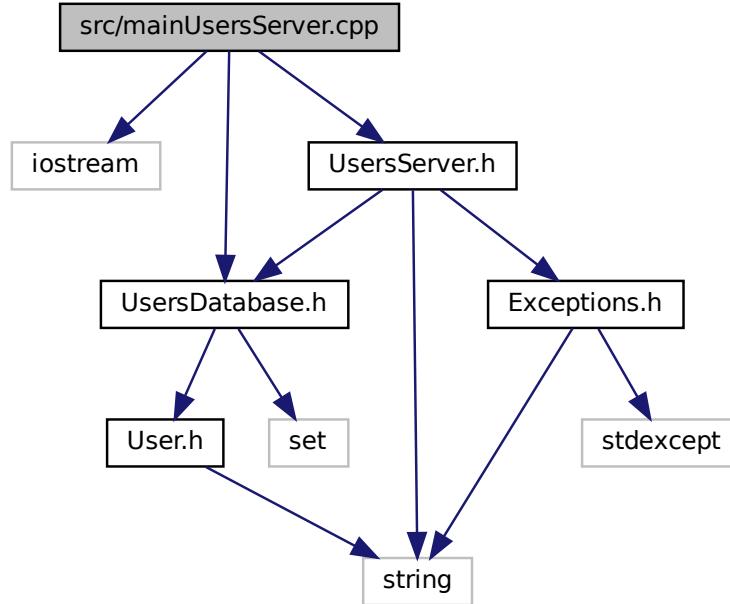
Here is the call graph for this function:



## 5.41 src/mainUsersServer.cpp File Reference

```
#include <iostream>
```

```
#include "UsersDatabase.h"
#include "UsersServer.h"
Include dependency graph for mainUsersServer.cpp:
```



## Functions

- int `main ()`

### 5.41.1 Function Documentation

#### 5.41.1.1 `main()`

```
int main ( )
```

Definition at line 7 of file `mainUsersServer.cpp`.

```

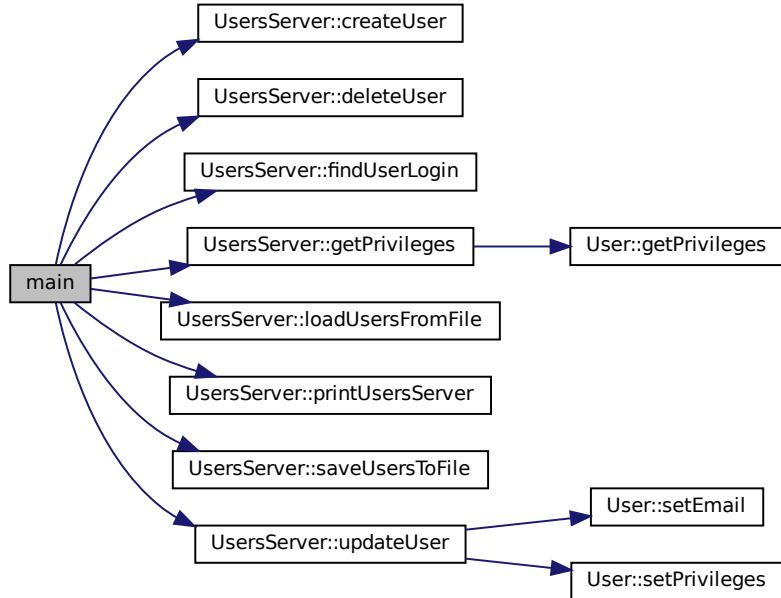
7   {
8     // creamos un server de usuarios
9     UsersServer usersServer;
10    // Imprimimos usuarios
11    usersServer.createUser("Lena", "LJ809K5ES43", "Bagumm.87;", "admin",
12      "liselese.ratte@aol.com");
13    usersServer.createUser("Lena", "LJ809K5ES43", "Bagumm.87;", "guest",
14      "liselese.ratte@aol.com");
15    cout << "*** Users created ***" << endl;
16    usersServer.printUsersServer();
17    cout << "-----" << endl;
18    usersServer.deleteUser("LJ809K5ES43");
19    usersServer.deleteUser("AJIJKIOKDDIJOIOJD");
20    usersServer.loadUsersFromFile();
```

```

21     usersServer.printUsersServer();
22     cout << "-----" << endl;
23     usersServer.createUser("adrian", "47552050X", "employee", "employee",
24     "adrian@example.com");
25     usersServer.saveUsersToFile();
26     if (usersServer.findUserLogin("adrian", "employee", "47552050X")) {
27         cout << "User found" << endl;
28     } else {
29         cout << "User not found" << endl;
30     }
31     if (usersServer.findUserLogin("AAA", "employee", "47552050X")) {
32         cout << "User found" << endl;
33     } else {
34         cout << "User not found" << endl;
35     }
36     cout << usersServer.getPrivileges("47552050X") << endl;
37     cout << usersServer.getPrivileges("475ffX") << endl;
38     usersServer.updateUser("adrian", "47552050X", "administrador", "admin",
39     "adrian3sAdminAhora");
40     usersServer.printUsersServer();
41     usersServer.saveUsersToFile();
42
43
44 */
45 UsersServer usersServer2;
46 usersServer2.loadUsersFromFile();
47 std::cout << "*** Users loaded from file (SERVER 2) ***" << std::endl;
48 usersServer2.printUsersServer();
49 */
50 }
```

References `UsersServer::createUser()`, `UsersServer::deleteUser()`, `UsersServer::findUserLogin()`, `UsersServer::getPrivileges()`, `UsersServer::loadUsersFromFile()`, `UsersServer::printUsersServer()`, `UsersServer::saveUsersToFile()`, and `UsersServer::updateUser()`.

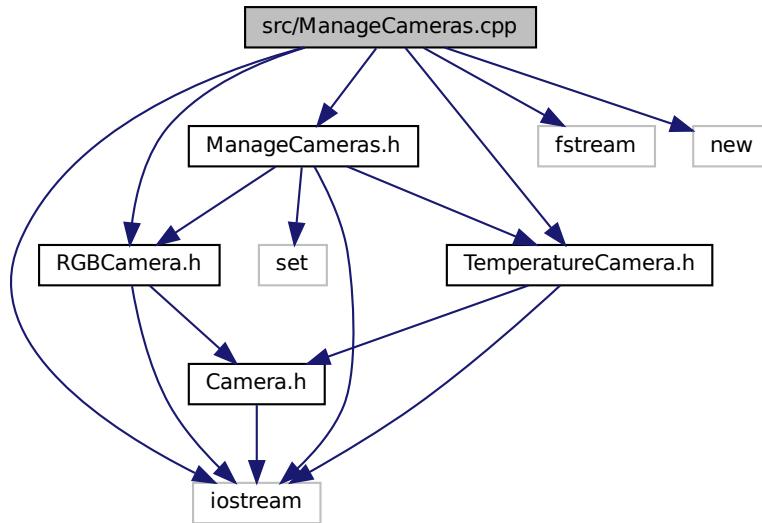
Here is the call graph for this function:



## 5.42 src/ManageCameras.cpp File Reference

```
#include "ManageCameras.h"
#include "RGBCamera.h"
```

```
#include "TemperatureCamera.h"
#include <fstream>
#include <iostream>
#include <new>
Include dependency graph for ManageCameras.cpp:
```

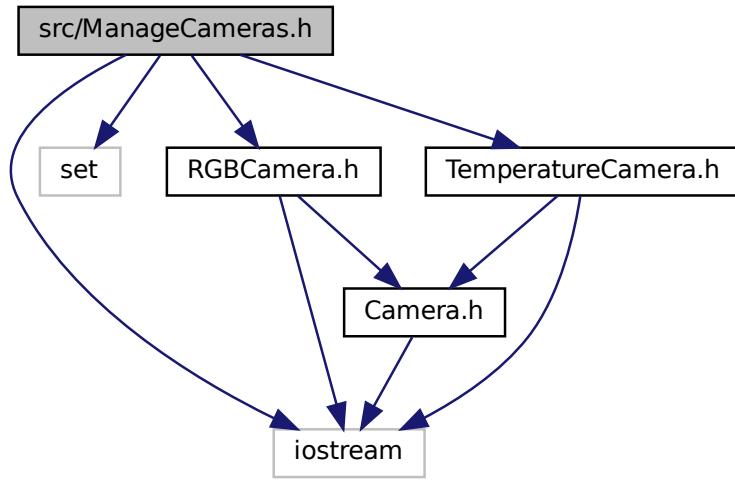


## 5.43 src/ManageCameras.h File Reference

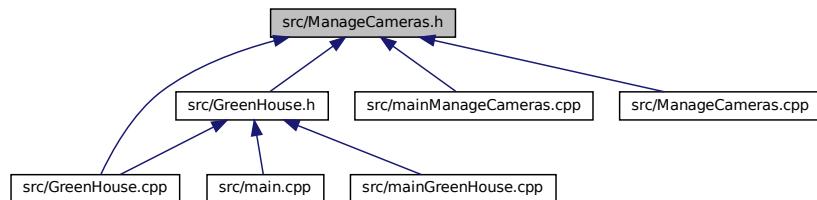
This is the class [ManageCameras](#). It contains the attributes and methods of the [ManageCameras](#) class. This class is used to manage the cameras of the system, it can create, remove, display, collect data, turn on/off the cameras and save/load the cameras to/from a file.

```
#include <iostream>
#include <set>
#include "RGBCamera.h"
#include "TemperatureCamera.h"
```

Include dependency graph for ManageCameras.h:



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



## Classes

- class [ManageCameras](#)

### 5.43.1 Detailed Description

This is the class [ManageCameras](#). It contains the attributes and methods of the `ManageCameras` class. This class is used to manage the cameras of the system, it can create, remove, display, collect data, turn on/off the cameras and save/load the cameras to/from a file.

#### Author

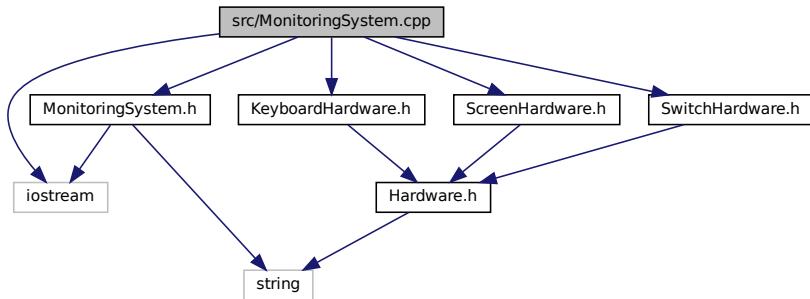
Adrián Montes Linares

#### Date

23/05/2024

## 5.44 src/MonitoringSystem.cpp File Reference

```
#include "MonitoringSystem.h"
#include <iostream>
#include "KeyboardHardware.h"
#include "ScreenHardware.h"
#include "SwitchHardware.h"
Include dependency graph for MonitoringSystem.cpp:
```



## Variables

- const int `MAIN_MENU_OPTIONS` = 2
- const int `ADMIN_MENU_OPTIONS` = 17
- const int `EMPLOYEE_MENU_OPTIONS` = 12
- const int `GUEST_MENU_OPTIONS` = 4

### 5.44.1 Variable Documentation

#### 5.44.1.1 ADMIN\_MENU\_OPTIONS

```
const int ADMIN_MENU_OPTIONS = 17
```

Definition at line 11 of file `MonitoringSystem.cpp`.

Referenced by `MonitoringSystem::mainWindowAdmin()`.

#### 5.44.1.2 EMPLOYEE\_MENU\_OPTIONS

```
const int EMPLOYEE_MENU_OPTIONS = 12
```

Definition at line 12 of file `MonitoringSystem.cpp`.

Referenced by `MonitoringSystem::mainWindowEmployee()`.

#### 5.44.1.3 GUEST\_MENU\_OPTIONS

```
const int GUEST_MENU_OPTIONS = 4
```

Definition at line 13 of file MonitoringSystem.cpp.

Referenced by MonitoringSystem::mainWindowGuest().

#### 5.44.1.4 MAIN\_MENU\_OPTIONS

```
const int MAIN_MENU_OPTIONS = 2
```

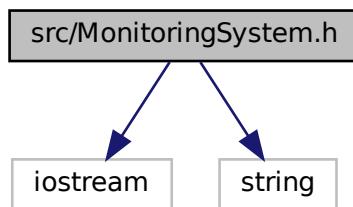
Definition at line 10 of file MonitoringSystem.cpp.

Referenced by MonitoringSystem::initialScreen().

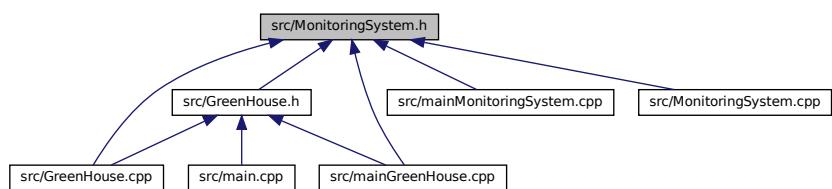
## 5.45 src/MonitoringSystem.h File Reference

This is the class [MonitoringSystem](#). It contains the attributes and methods of the [MonitoringSystem](#) class, this class.

```
#include <iostream>
#include <string>
Include dependency graph for MonitoringSystem.h:
```



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



## Classes

- class [MonitoringSystem](#)

## Variables

- const int [MAIN\\_MENU\\_OPTIONS](#)
- const int [ADMIN\\_MENU\\_OPTIONS](#)
- const int [EMPLOYEE\\_MENU\\_OPTIONS](#)
- const int [GUEST\\_MENU\\_OPTIONS](#)

### 5.45.1 Detailed Description

This is the class [MonitoringSystem](#). It contains the attributes and methods of the [MonitoringSystem](#) class, this class.

#### Author

Adrián Montes Linares

#### Date

21/04/2024

### 5.45.2 Variable Documentation

#### 5.45.2.1 ADMIN\_MENU\_OPTIONS

```
const int ADMIN_MENU_OPTIONS [extern]
```

Definition at line 11 of file MonitoringSystem.cpp.

Referenced by [MonitoringSystem::mainWindowAdmin\(\)](#).

#### 5.45.2.2 EMPLOYEE\_MENU\_OPTIONS

```
const int EMPLOYEE_MENU_OPTIONS [extern]
```

Definition at line 12 of file MonitoringSystem.cpp.

Referenced by [MonitoringSystem::mainWindowEmployee\(\)](#).

#### 5.45.2.3 GUEST\_MENU\_OPTIONS

```
const int GUEST_MENU_OPTIONS [extern]
```

Definition at line 13 of file MonitoringSystem.cpp.

Referenced by MonitoringSystem::mainWindowGuest().

#### 5.45.2.4 MAIN\_MENU\_OPTIONS

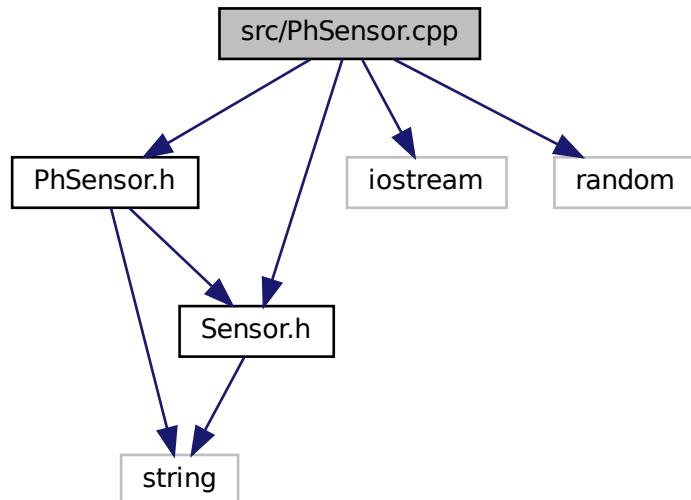
```
const int MAIN_MENU_OPTIONS [extern]
```

Definition at line 10 of file MonitoringSystem.cpp.

Referenced by MonitoringSystem::initialScreen().

## 5.46 src/PhSensor.cpp File Reference

```
#include "PhSensor.h"
#include <iostream>
#include <random>
#include "Sensor.h"
Include dependency graph for PhSensor.cpp:
```



## Functions

- std::ostream & operator<< (std::ostream &os, const PhSensor &sensor)

## 5.46.1 Function Documentation

### 5.46.1.1 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const PhSensor & sensor )
```

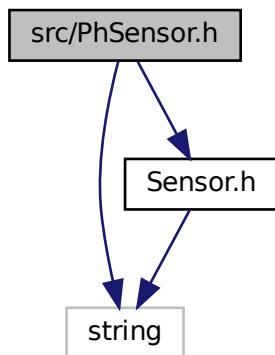
Definition at line 55 of file PhSensor.cpp.

```
55
56     sensor.printData();
57     return os;
58 }
```

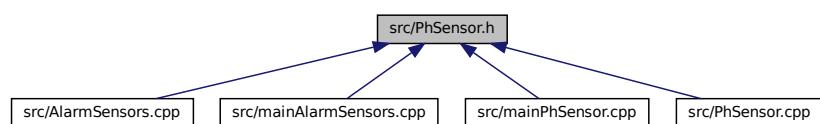
## 5.47 src/PhSensor.h File Reference

This is the class [PhSensor](#). It contains the attributes and methods of the [PhSensor](#) class.

```
#include <string>
#include "Sensor.h"
Include dependency graph for PhSensor.h:
```



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



## Classes

- class [PhSensor](#)

### 5.47.1 Detailed Description

This is the class [PhSensor](#). It contains the attributes and methods of the [PhSensor](#) class.

#### Author

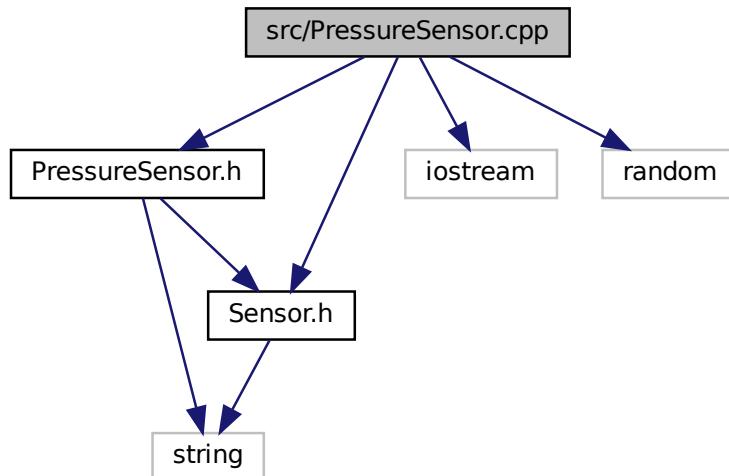
Adrián Montes Linares

#### Date

21/04/2024

## 5.48 src/PressureSensor.cpp File Reference

```
#include "PressureSensor.h"
#include <iostream>
#include <random>
#include "Sensor.h"
Include dependency graph for PressureSensor.cpp:
```



## Functions

- std::ostream & [operator<<](#) (std::ostream &os, const [PressureSensor](#) &sensor)

## 5.48.1 Function Documentation

### 5.48.1.1 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const PressureSensor & sensor )
```

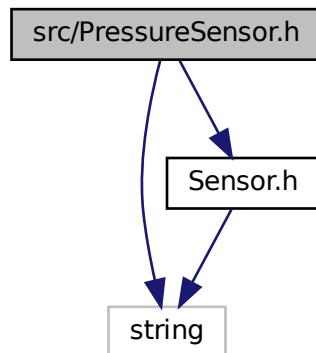
Definition at line 32 of file PressureSensor.cpp.

```
32
33     sensor.printData();
34     return os;
35 }
```

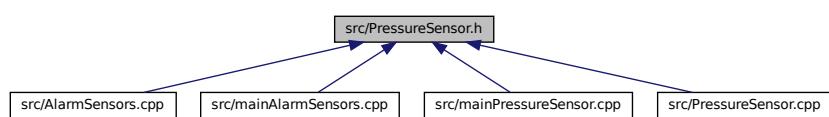
## 5.49 src/PressureSensor.h File Reference

This is the class [PressureSensor](#). It contains the attributes and methods of the [PressureSensor](#) class.

```
#include <string>
#include "Sensor.h"
Include dependency graph for PressureSensor.h:
```



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



## Classes

- class PressureSensor

### 5.49.1 Detailed Description

This is the class PressureSensor. It contains the attributes and methods of the PressureSensor class.

#### Author

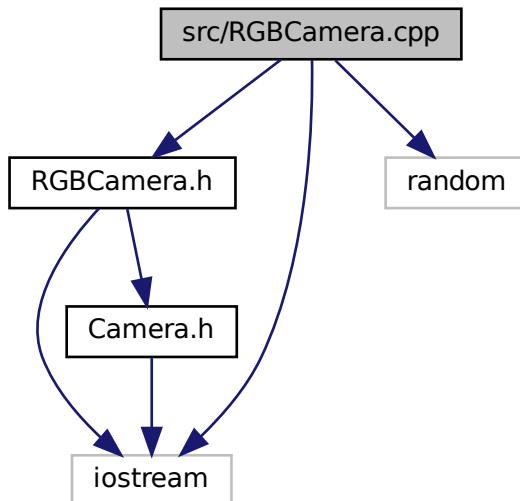
Adrián Montes Linares

#### Date

21/04/2024

## 5.50 src/RGBCamera.cpp File Reference

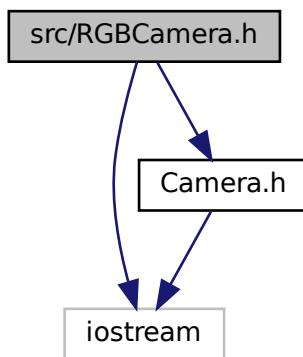
```
#include "RGBCamera.h"
#include <iostream>
#include <random>
Include dependency graph for RGBCamera.cpp:
```



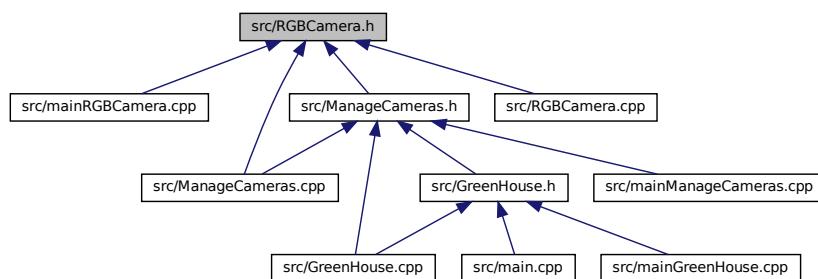
## 5.51 src/RGBCamera.h File Reference

This is the class [RGBCamera](#). It contains the attributes and methods of the [RGBCamera](#) class. This class is used to represent a RGB camera of the system, it can collect data, turn on/off the camera, print the camera information and set the RGB values of the camera.

```
#include <iostream>
#include "Camera.h"
Include dependency graph for RGBCamera.h:
```



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



### Classes

- class [RGBCamera](#)

#### 5.51.1 Detailed Description

This is the class [RGBCamera](#). It contains the attributes and methods of the [RGBCamera](#) class. This class is used to represent a RGB camera of the system, it can collect data, turn on/off the camera, print the camera information and set the RGB values of the camera.

**Author**

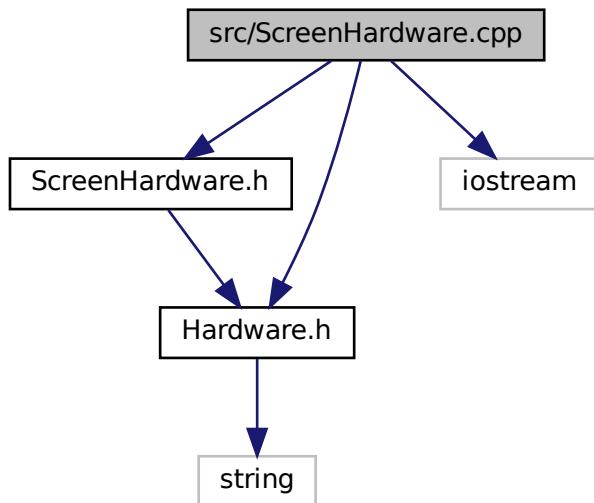
Adrián Montes Linares

**Date**

23/05/2024

## 5.52 src/ScreenHardware.cpp File Reference

```
#include "ScreenHardware.h"
#include <iostream>
#include "Hardware.h"
Include dependency graph for ScreenHardware.cpp:
```



## Variables

- const std::string [USER\\_PROMPT](#)
- const std::string [ASK\\_DATA](#) = "Please enter all the data required"

### 5.52.1 Variable Documentation

### 5.52.1.1 ASK\_DATA

```
const std::string ASK_DATA = "Please enter all the data required"
```

Definition at line 13 of file ScreenHardware.cpp.

Referenced by ScreenHardware::createUserWindow(), ScreenHardware::loginWindow(), and ScreenHardware::updateUserWindow().

### 5.52.1.2 USER\_PROMPT

```
const std::string USER_PROMPT
```

**Initial value:**

```
=
    "First the name(intro), then the new password(intro), then the "
    "nif(intro), then the new role(intro), then the status(intro), "
    "then the email(intro)"
```

Definition at line 8 of file ScreenHardware.cpp.

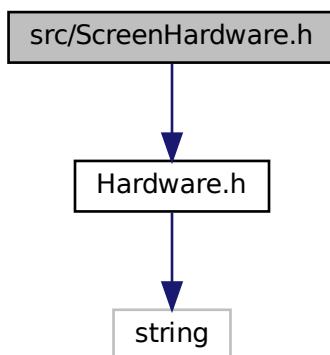
Referenced by ScreenHardware::createUserWindow(), and ScreenHardware::updateUserWindow().

## 5.53 src/ScreenHardware.h File Reference

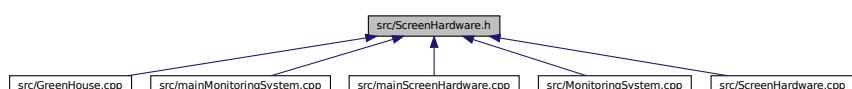
This is the class [ScreenHardware](#). It contains the attributes and methods of the [ScreenHardware](#) class, this class is a child of the [Hardware](#) class. This class is used to display the output of the system and ask for an input before with the keyboard.

```
#include "Hardware.h"
```

Include dependency graph for ScreenHardware.h:



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



## Classes

- class [ScreenHardware](#)

### 5.53.1 Detailed Description

This is the class [ScreenHardware](#). It contains the attributes and methods of the [ScreenHardware](#) class, this class is a child of the [Hardware](#) class. This class is used to display the output of the system and ask for an input before with the keyboard.

#### Author

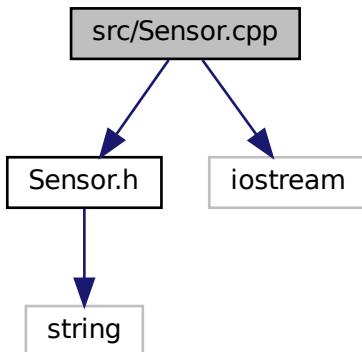
Adrián Montes Linares

#### Date

21/04/2024

## 5.54 src/Sensor.cpp File Reference

```
#include "Sensor.h"
#include <iostream>
Include dependency graph for Sensor.cpp:
```



## Functions

- std::ostream & [operator<<](#) (std::ostream &os, const [Sensor](#) &[Sensor](#))
- std::istream & [operator>>](#) (std::istream &is, [Sensor](#) &[sensor](#))

### 5.54.1 Function Documentation

### 5.54.1.1 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const Sensor & Sensor )
```

**Parameters**

<i>os</i>	
<a href="#">Sensor</a>	

**Returns**

std::ostream&amp;

Definition at line 102 of file Sensor.cpp.

```

102
103     os << "ID: " << Sensor.getId() << " Type: " << Sensor.getType()
104     << " Active: " << Sensor.isActive() << " Data: " << Sensor.getData()
105     << std::endl;
106     return os;
107 }
```

**5.54.1.2 operator>>()**

```
std::istream& operator>> (
    std::istream & is,
    Sensor & sensor )
```

**Parameters**

<i>is</i>	
<a href="#">Sensor</a>	

**Returns**

std::istream&amp;

Definition at line 109 of file Sensor.cpp.

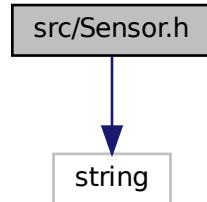
```

109
110     cout << "Enter sensor ID: ";
111     is >> sensor.id_;
112     cout << "Enter the type: ";
113     std::string type;
114     is >> type;
115     sensor.setType(type);
116     cout << "Enter sensor active: ";
117     is >> sensor.active_;
118
119     return is;
120 }
```

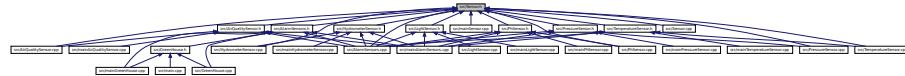
**5.55 src/Sensor.h File Reference**

This is the class [Sensor](#). It contains the attributes and methods of the [Sensor](#) class.

```
#include <string>
Include dependency graph for Sensor.h:
```



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



## Classes

- class [Sensor](#)

### 5.55.1 Detailed Description

This is the class [Sensor](#). It contains the attributes and methods of the [Sensor](#) class.

#### Author

Adrián Montes Linares

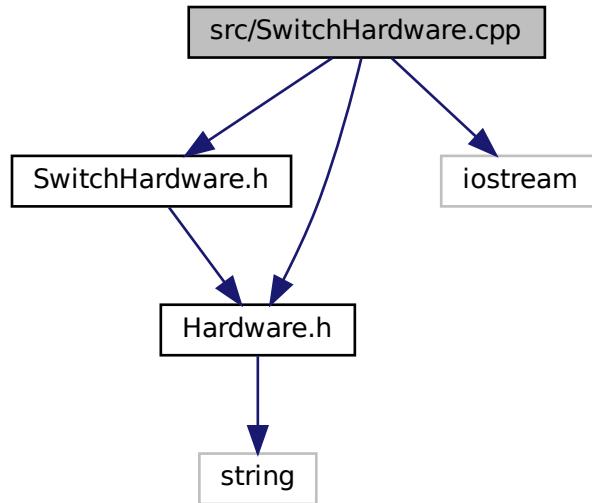
#### Date

21/04/2024

## 5.56 src/SwitchHardware.cpp File Reference

```
#include "SwitchHardware.h"
#include <iostream>
```

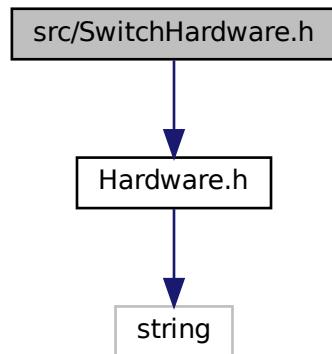
```
#include "Hardware.h"  
Include dependency graph for SwitchHardware.cpp:
```



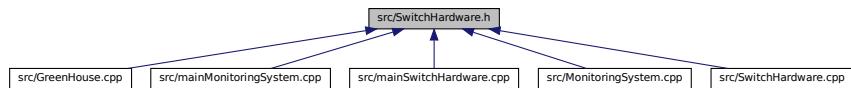
## 5.57 src/SwitchHardware.h File Reference

This is the class [SwitchHardware](#). It contains the attributes and methods of the [SwitchHardware](#) class, this class is a child of the [Hardware](#) class.

```
#include "Hardware.h"  
Include dependency graph for SwitchHardware.h:
```



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



## Classes

- class [SwitchHardware](#)

### 5.57.1 Detailed Description

This is the class [SwitchHardware](#). It contains the attributes and methods of the [SwitchHardware](#) class, this class is a child of the [Hardware](#) class.

#### Author

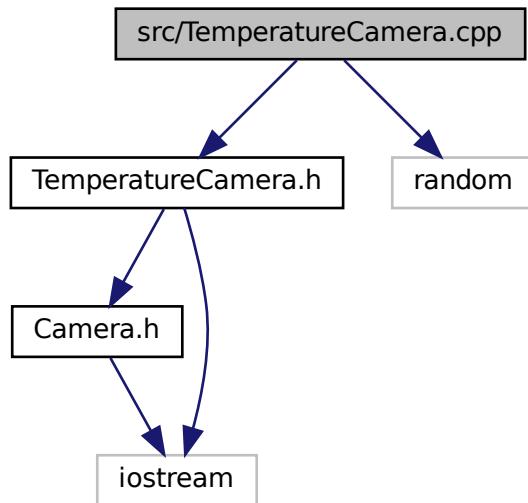
Adrián Montes Linares

#### Date

21/04/2024

## 5.58 src/TemperatureCamera.cpp File Reference

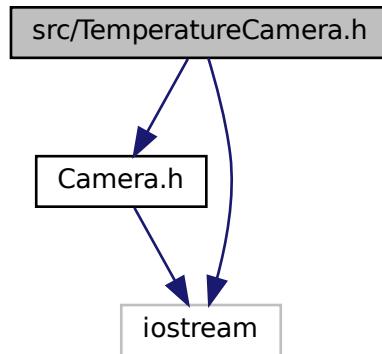
```
#include "TemperatureCamera.h"
#include <random>
Include dependency graph for TemperatureCamera.cpp:
```



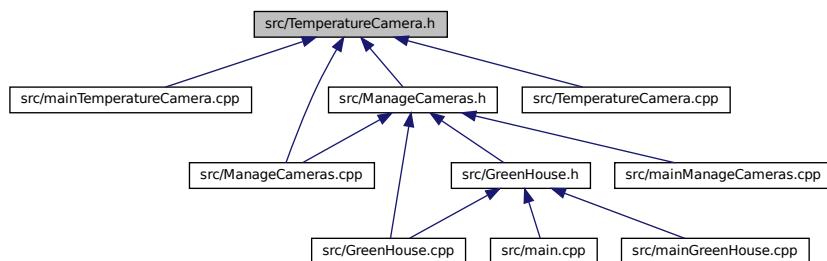
## 5.59 src/TemperatureCamera.h File Reference

This is the class Temperature Camera. It contains the attributes and methods of the [TemperatureCamera](#) class. This class is used to represent a temperature camera of the system, it can collect data, print the camera information and set the temperature of the camera.

```
#include "Camera.h"
#include <iostream>
Include dependency graph for TemperatureCamera.h:
```



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



## Classes

- class [TemperatureCamera](#)

### 5.59.1 Detailed Description

This is the class Temperature Camera. It contains the attributes and methods of the [TemperatureCamera](#) class. This class is used to represent a temperature camera of the system, it can collect data, print the camera information and set the temperature of the camera.

**Author**

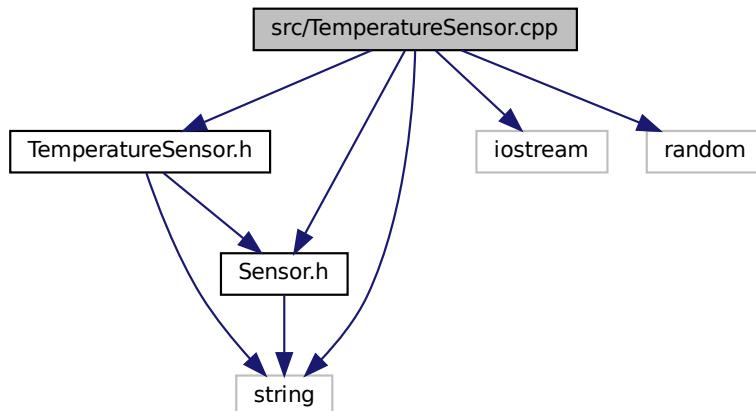
Adrián Montes Linares

**Date**

23/05/2024

## 5.60 src/TemperatureSensor.cpp File Reference

```
#include "TemperatureSensor.h"
#include <iostream>
#include <random>
#include <string>
#include "Sensor.h"
Include dependency graph for TemperatureSensor.cpp:
```



## Functions

- std::ostream & `operator<<` (std::ostream &os, const TemperatureSensor &sensor)

### 5.60.1 Function Documentation

#### 5.60.1.1 `operator<<()`

```
std::ostream& operator<< (
    std::ostream & os,
    const TemperatureSensor & sensor )
```

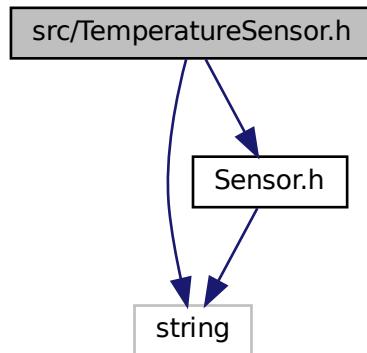
Definition at line 36 of file TemperatureSensor.cpp.

```
36
37     sensor.printData();
38     return os;
39 }
```

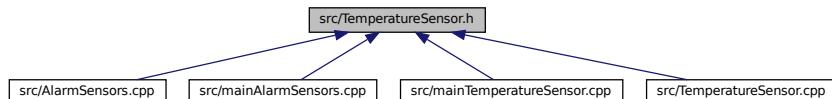
## 5.61 src/TemperatureSensor.h File Reference

This is the class [TemperatureSensor](#). It contains the attributes and methods of the [TemperatureSensor](#) class.

```
#include <string>
#include "Sensor.h"
Include dependency graph for TemperatureSensor.h:
```



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



### Classes

- class [TemperatureSensor](#)

#### 5.61.1 Detailed Description

This is the class [TemperatureSensor](#). It contains the attributes and methods of the [TemperatureSensor](#) class.

##### Author

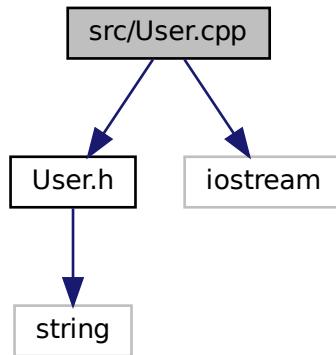
Adrián Montes Linares

##### Date

21/04/2024

## 5.62 src/User.cpp File Reference

```
#include "User.h"
#include <iostream>
Include dependency graph for User.cpp:
```



### Functions

- std::ostream & `operator<<` (std::ostream &os, const `User` &user)
- std::istream & `operator>>` (std::istream &is, `User` &user)

#### 5.62.1 Function Documentation

##### 5.62.1.1 `operator<<()`

```
std::ostream& operator<< (
    std::ostream & os,
    const User & user )
```

###### Parameters

<code>os</code>	
<code>user</code>	

###### Returns

```
std::ostream&
```

Definition at line 76 of file User.cpp.

```

76     os << user.getName() << " " << user.getNif() << " " << user.getPassword()
77     << " " << user.getPrivileges() << " " << user.getEmail() << std::endl;
78
79     return os;
80 }
```

### 5.62.1.2 operator>>()

```
std::istream& operator>> (
    std::istream & is,
    User & user )
```

#### Parameters

<i>is</i>	
<i>user</i>	

#### Returns

std::istream&

Definition at line 83 of file User.cpp.

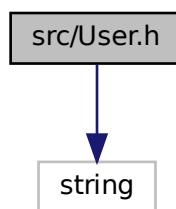
```

83
84     std::string privilege;
85     is >> user.name >> user.nif >> user.password >> privilege >> user.email;
86     user.setPrivileges(privilege);
87     return is;
88 }
```

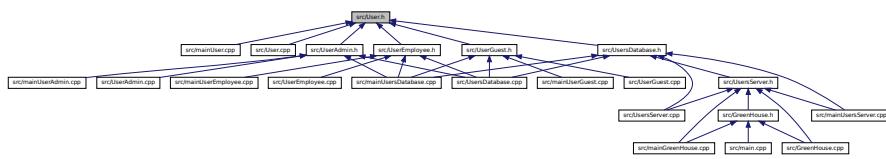
## 5.63 src/User.h File Reference

This is the class [User](#). It contains the attributes and methods of the [User](#) class.

```
#include <string>
Include dependency graph for User.h:
```



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



## Classes

- class [User](#)

### 5.63.1 Detailed Description

This is the class [User](#). It contains the attributes and methods of the [User](#) class.

#### Author

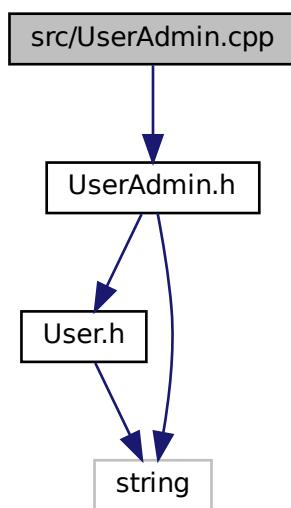
Adrián Montes Linares

#### Date

21/04/2024

## 5.64 src/UserAdmin.cpp File Reference

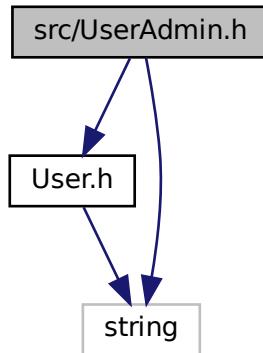
```
#include "UserAdmin.h"
Include dependency graph for UserAdmin.cpp:
```



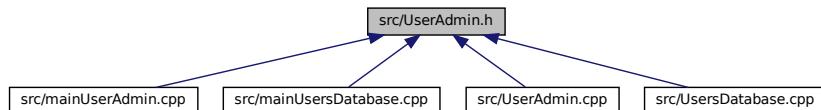
## 5.65 src/UserAdmin.h File Reference

This is the class [UserAdmin](#). It contains the attributes and methods of the [UserAdmin](#) class.

```
#include "User.h"
#include <string>
Include dependency graph for UserAdmin.h:
```



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



### Classes

- class [UserAdmin](#)

#### 5.65.1 Detailed Description

This is the class [UserAdmin](#). It contains the attributes and methods of the [UserAdmin](#) class.

##### Author

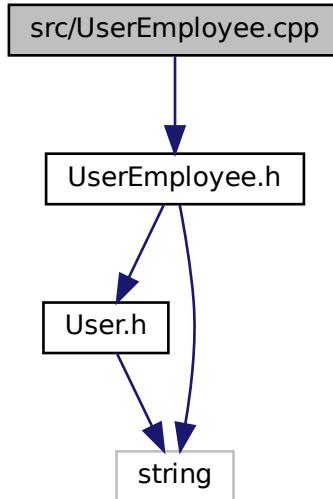
Adrián Montes Linares

##### Date

21/04/2024

## 5.66 src/UserEmployee.cpp File Reference

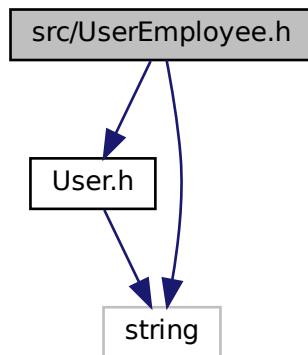
```
#include "UserEmployee.h"  
Include dependency graph for UserEmployee.cpp:
```



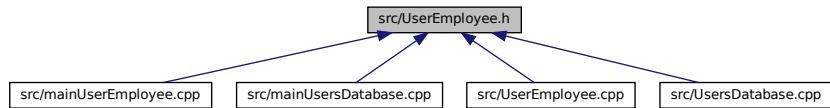
## 5.67 src/UserEmployee.h File Reference

This is the class [UserEmployee](#). It contains the attributes and methods of the [UserEmployee](#) class.

```
#include "User.h"  
#include <string>  
Include dependency graph for UserEmployee.h:
```



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



## Classes

- class [UserEmployee](#)

### 5.67.1 Detailed Description

This is the class [UserEmployee](#). It contains the attributes and methods of the [UserEmployee](#) class.

#### Author

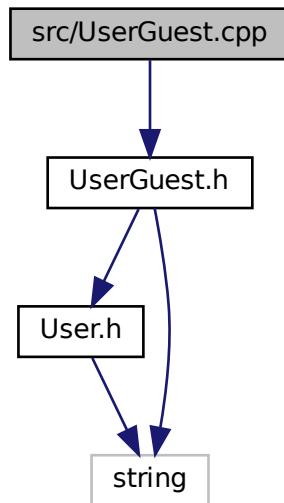
Adrián Montes Linares

#### Date

21/04/2024

## 5.68 src/UserGuest.cpp File Reference

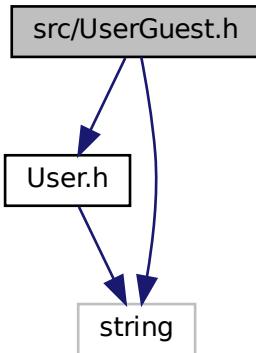
```
#include "UserGuest.h"
Include dependency graph for UserGuest.cpp:
```



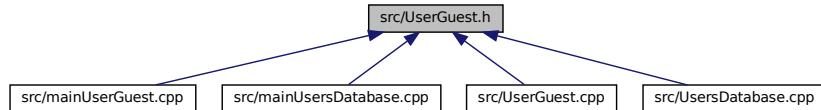
## 5.69 src/UserGuest.h File Reference

This is the class [UserGuest](#). It contains the attributes and methods of the [UserGuest](#) class.

```
#include "User.h"
#include <string>
Include dependency graph for UserGuest.h:
```



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



### Classes

- class [UserGuest](#)

#### 5.69.1 Detailed Description

This is the class [UserGuest](#). It contains the attributes and methods of the [UserGuest](#) class.

##### Author

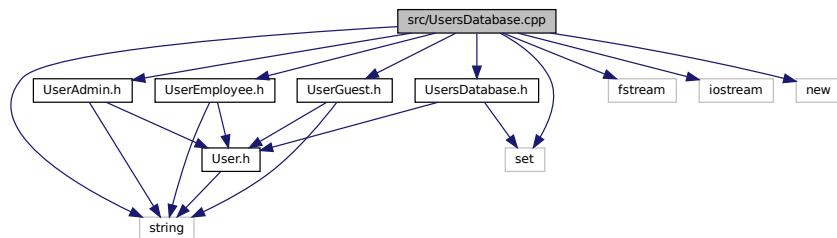
Adrián Montes Linares

##### Date

21/04/2024

## 5.70 src/UsersDatabase.cpp File Reference

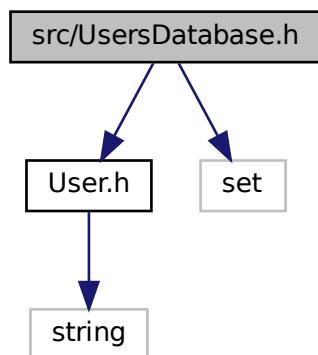
```
#include "UsersDatabase.h"
#include <fstream>
#include <iostream>
#include <new>
#include <set>
#include <string>
#include "UserAdmin.h"
#include "UserEmployee.h"
#include "UserGuest.h"
Include dependency graph for UsersDatabase.cpp:
```



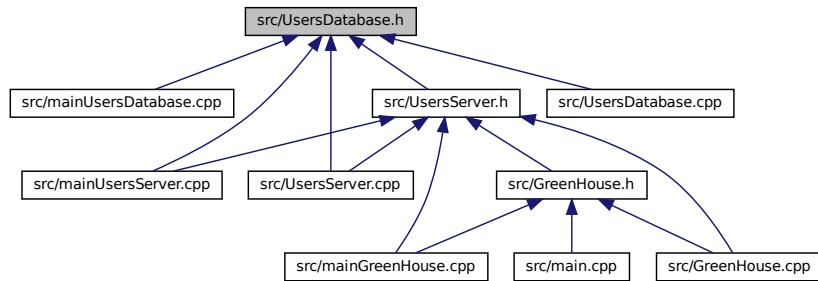
## 5.71 src/UsersDatabase.h File Reference

This is the class [UsersDatabase](#). It contains the attributes and methods of the [UsersDatabase](#) class.

```
#include "User.h"
#include <set>
Include dependency graph for UsersDatabase.h:
```



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



## Classes

- class [UserPtrComparator](#)
- class [UserNameComparator](#)
- class [UsersDatabase](#)

### 5.71.1 Detailed Description

This is the class [UsersDatabase](#). It contains the attributes and methods of the [UsersDatabase](#) class.

#### Author

Adrián Montes Linares

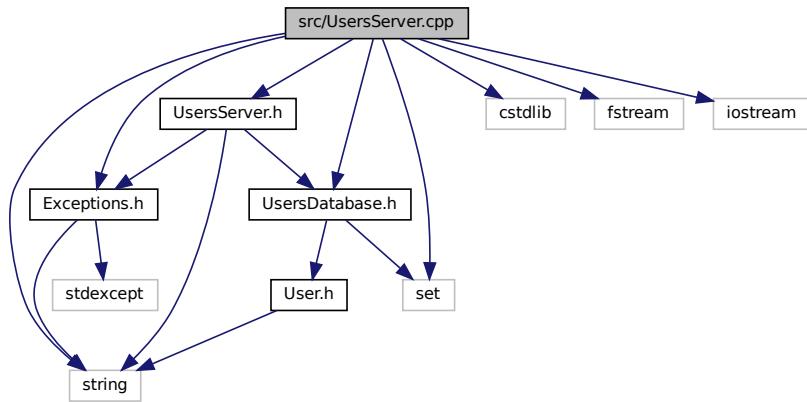
#### Date

21/04/2024

## 5.72 src/UsersServer.cpp File Reference

```
#include "UsersServer.h"
#include <cstdlib>
#include <fstream>
#include <iostream>
#include <string>
#include <set>
#include "Exceptions.h"
```

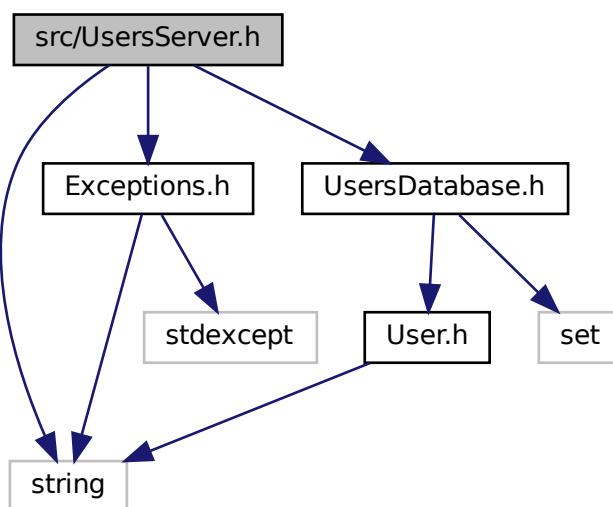
```
#include "UsersDatabase.h"  
Include dependency graph for UsersServer.cpp:
```



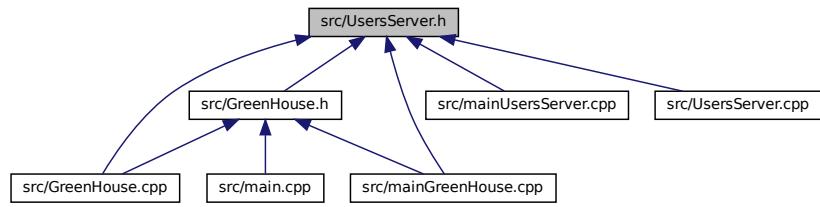
## 5.73 src/UsersServer.h File Reference

This is the class [UsersServer](#). It contains the attributes and methods of the [UsersServer](#) class.

```
#include <string>  
#include "Exceptions.h"  
#include "UsersDatabase.h"  
Include dependency graph for UsersServer.h:
```



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



## Classes

- class [UsersServer](#)

### 5.73.1 Detailed Description

This is the class [UsersServer](#). It contains the attributes and methods of the [UsersServer](#) class.

#### Author

Adrián Montes Linares

#### Date

21/04/2024

# Index

~AirQualitySensor  
    AirQualitySensor, 12  
~AlarmSensors  
    AlarmSensors, 20  
~Camera  
    Camera, 32  
~GreenHouse  
    GreenHouse, 53  
~Hardware  
    Hardware, 75  
~HydrometerSensor  
    HydrometerSensor, 84  
~KeyboardHardware  
    KeyboardHardware, 92  
~LightSensor  
    LightSensor, 98  
~ManageCameras  
    ManageCameras, 105  
~MonitoringSystem  
    MonitoringSystem, 119  
~PhSensor  
    PhSensor, 164  
~PressureSensor  
    PressureSensor, 173  
~RGBCamera  
    RGBCamera, 182  
~ScreenHardware  
    ScreenHardware, 194  
~Sensor  
    Sensor, 212  
~SwitchHardware  
    SwitchHardware, 228  
~TemperatureCamera  
    TemperatureCamera, 235  
~TemperatureSensor  
    TemperatureSensor, 242  
~User  
    User, 252  
~UserAdmin  
    UserAdmin, 268  
~UserEmployee  
    UserEmployee, 272  
~UserGuest  
    UserGuest, 276  
~UsersDatabase  
    UsersDatabase, 281  
~UsersServer  
    UsersServer, 298  
  
active\_  
  
    Camera, 37  
    Hardware, 81  
    Sensor, 225  
addCamera  
    ManageCameras, 105  
addSensor  
    AlarmSensors, 20  
addUser  
    UsersDatabase, 281  
ADMIN\_MENU\_OPTIONS  
    MonitoringSystem.cpp, 370  
    MonitoringSystem.h, 372  
AIR\_QUALITY  
    Sensor, 211  
AirQualitySensor, 9  
    ~AirQualitySensor, 12  
    AirQualitySensor, 12  
    checkAllgood, 13  
    collectAndPrint, 13  
    collectData, 14  
    operator<<, 17  
    printData, 15  
    stringStatus, 16  
AirQualitySensor.cpp  
    operator<<, 311  
alarm\_  
    GreenHouse, 70  
AlarmSensors, 18  
    ~AlarmSensors, 20  
    addSensor, 20  
    AlarmSensors, 19  
    checkAllgood, 21  
    checkSensors, 21  
    deleteSensor, 22  
    displayAlarmStatus, 23  
    displayAllSensorsData, 23  
    fileNameBin, 29  
    fileNameTxt, 29  
    loadSensorsData, 24  
    loadSensorsDataBin, 24  
    loadSensorsDataTxt, 25  
    saveSensorsData, 25  
    saveSensorsDataBin, 26  
    saveSensorsDataTxt, 26  
    sensorExists, 26  
    sensors\_, 29  
    sensorsInitialized, 27  
    status\_, 29  
    turnOffSystem, 27

turnOnOffSystem, 28  
 turnOnSystem, 28  
**ASK\_DATA**  
 ScreenHardware.cpp, 379  
**askEmail**  
 MonitoringSystem, 119  
**askIdAlarm**  
 MonitoringSystem, 120  
**askIdCamera**  
 MonitoringSystem, 121  
**askInput**  
 Hardware, 75  
 KeyboardHardware, 93  
 ScreenHardware, 194  
 SwitchHardware, 228  
**askInputInt**  
 MonitoringSystem, 121  
**askMainUser**  
 MonitoringSystem, 122  
**askName**  
 MonitoringSystem, 123  
**askNIF**  
 MonitoringSystem, 124  
**askPassword**  
 MonitoringSystem, 125  
**askPrivileges**  
 MonitoringSystem, 126  
**askTypeAlarm**  
 MonitoringSystem, 126  
**askTypeCamera**  
 MonitoringSystem, 127  
  
**blue\_**  
 RGBCamera, 189  
  
**Camera**, 30  
 ~Camera, 32  
 active\_, 37  
 Camera, 32  
 collectData, 33  
 getId, 33  
 getType, 33  
 id\_, 37  
 isActive, 34  
 printCamera, 34  
 setId, 34  
 setType, 36  
 turnOff, 36  
 turnOn, 36  
 type\_, 37  
**cameras\_**  
 ManageCameras, 113  
**checkAllgood**  
 AirQualitySensor, 13  
 AlarmSensors, 21  
 HydrometerSensor, 85  
 LightSensor, 99  
 PhSensor, 165  
 PressureSensor, 174  
  
 Sensor, 212  
 TemperatureSensor, 243  
**checkSensors**  
 AlarmSensors, 21  
**cleanScreen**  
 MonitoringSystem, 128  
 ScreenHardware, 195  
**clearCameras**  
 ManageCameras, 106  
**collectAndDisplayData**  
 ManageCameras, 106  
**collectAndPrint**  
 AirQualitySensor, 13  
 HydrometerSensor, 85  
 LightSensor, 99  
 PhSensor, 165  
 PressureSensor, 174  
 Sensor, 213  
 TemperatureSensor, 243  
**collectData**  
 AirQualitySensor, 14  
 Camera, 33  
 HydrometerSensor, 86  
 LightSensor, 100  
 ManageCameras, 106  
 PhSensor, 166  
 PressureSensor, 175  
 RGBCamera, 182  
 Sensor, 213  
 TemperatureCamera, 235  
 TemperatureSensor, 244  
**createAlarmScreen**  
 MonitoringSystem, 129  
**createAlarmWindow**  
 ScreenHardware, 196  
**createCamera**  
 ManageCameras, 107  
**createCameraScreen**  
 MonitoringSystem, 130  
**createCameraWindow**  
 ScreenHardware, 196  
**createUser**  
 UsersServer, 298  
**createUserScreen**  
 MonitoringSystem, 130  
**createUserWindow**  
 ScreenHardware, 197  
  
**data\_**  
 KeyboardHardware, 95  
 Sensor, 225  
**deleteAlarmScreen**  
 MonitoringSystem, 131  
**deleteAlarmWindow**  
 ScreenHardware, 197  
**deleteCameraScreen**  
 MonitoringSystem, 132  
**deleteCameraWindow**  
 ScreenHardware, 198

deleteSensor  
    AlarmSensors, 22  
deleteUser  
    UsersDatabase, 282  
    UsersServer, 299  
deleteUserByEmail  
    UsersDatabase, 283  
deleteUserByName  
    UsersDatabase, 284  
deleteUserByNif  
    UsersDatabase, 285  
deleteUserScreen  
    MonitoringSystem, 133  
deleteUserWindow  
    ScreenHardware, 198  
displayAlarmsScreen  
    MonitoringSystem, 133  
displayAlarmStatus  
    AlarmSensors, 23  
displayAlarmsWindow  
    ScreenHardware, 199  
displayAllCameras  
    ManageCameras, 108  
displayAllSensorsData  
    AlarmSensors, 23  
displayCameraScreen  
    MonitoringSystem, 134  
displayCameraWindow  
    ScreenHardware, 199  
displayErrorScreen  
    MonitoringSystem, 135  
displayErrorWindow  
    ScreenHardware, 200  
displayOutput  
    Hardware, 75  
    KeyboardHardware, 93  
    ScreenHardware, 200  
    SwitchHardware, 229  
displaySensorsScreen  
    MonitoringSystem, 136  
displaySensorsWindow  
    ScreenHardware, 201  
displayUsersScreen  
    MonitoringSystem, 137  
displayUsersWindow  
    ScreenHardware, 201  
  
email  
    User, 263  
emailSelectedUser\_  
    MonitoringSystem, 157  
EMPLOYEE\_MENU\_OPTIONS  
    MonitoringSystem.cpp, 370  
    MonitoringSystem.h, 372  
exitScreen  
    MonitoringSystem, 137  
exitWindow  
    ScreenHardware, 201  
  
FileCloseError, 38  
    FileCloseError, 39  
FileCorruptError, 39  
    FileCorruptError, 40  
FileLockError, 41  
    FileLockError, 42  
filename\_  
    ManageCameras, 114  
fileNameBi\_  
    UsersServer, 308  
fileNameBin  
    AlarmSensors, 29  
fileNameTxt  
    AlarmSensors, 29  
fileNameTxt\_  
    UsersServer, 308  
FileNotFoundException, 42  
    FileNotFoundException, 44  
FileOpenError, 44  
    FileOpenError, 45  
FilePermissionError, 46  
    FilePermissionError, 47  
FileReadError, 47  
    FileReadError, 49  
FileWriteError, 49  
    FileWriteError, 50  
findCamera  
    ManageCameras, 108  
findUser  
    UsersDatabase, 286  
findUserByEmail  
    UsersDatabase, 287  
findUserByName  
    UsersDatabase, 288  
findUserByNif  
    UsersDatabase, 289  
findUser>Password  
    UsersDatabase, 290  
findUserLogin  
    UsersServer, 299  
  
getBlue  
    RGBCamera, 183  
getData  
    Sensor, 214  
getEmail  
    User, 252  
getEmailSelectedUser  
    MonitoringSystem, 138  
getGreen  
    RGBCamera, 183  
getId  
    Camera, 33  
    Sensor, 215  
getIdAlarm  
    MonitoringSystem, 139  
getIdCamera  
    MonitoringSystem, 139  
getIdSelectedAlarm

MonitoringSystem, 140  
 getIdSelectedCamera  
     MonitoringSystem, 140  
 getName  
     MonitoringSystem, 141  
     User, 253  
 getNameSelectedUser  
     MonitoringSystem, 141  
 getNIF  
     MonitoringSystem, 142  
 getNif  
     User, 254  
 getNIFSelectedUser  
     MonitoringSystem, 142  
 getPassword  
     MonitoringSystem, 143  
     User, 254  
 getPasswordSelectedUser  
     MonitoringSystem, 143  
 getPrivileges  
     User, 255  
     UsersServer, 300  
 getPrivilegesSelectedUser  
     MonitoringSystem, 144  
 getRed  
     RGBCamera, 184  
 getSelection  
     MonitoringSystem, 144  
 getTemperature  
     TemperatureCamera, 236  
 getType  
     Camera, 33  
     Hardware, 76  
     Sensor, 216  
 getTypeAlarm  
     MonitoringSystem, 145  
 getTypeCamera  
     MonitoringSystem, 145  
 getUsers  
     UsersDatabase, 290  
 getUsersSortedByName  
     UsersDatabase, 291  
 green\_  
     RGBCamera, 189  
 GreenHouse, 51  
     ~GreenHouse, 53  
     alarm\_, 70  
     GreenHouse, 53  
     mainSystem, 53  
     manageAdmin, 55  
     manageCreateUser, 59  
     manageDeleteUser, 59  
     manageEmployee, 60  
     manageGuest, 63  
     manageLogin, 64  
     manageUpdateUser, 67  
     mc\_, 70  
     ms\_, 71  
     save, 68  
     searchUser, 68  
     sleep, 69  
     startSystem, 69  
     us\_, 71  
 GUEST\_MENU\_OPTIONS  
     MonitoringSystem.cpp, 370  
     MonitoringSystem.h, 372  
 Hardware, 72  
     ~Hardware, 75  
     active\_, 81  
     askInput, 75  
     displayOutput, 75  
     getType, 76  
     Hardware, 74, 75  
     isActive, 76  
     KEYBOARD, 74  
     NONE, 74  
     SCREEN, 74  
     setType, 77  
     stringStatus, 78  
     stringToType, 79  
     SWITCH, 74  
     turnOff, 80  
     turnOn, 80  
     type\_, 81  
     Types\_Hardware, 74  
     typeToString, 80  
 HYDROMETER  
     Sensor, 211  
 HydrometerSensor, 82  
     ~HydrometerSensor, 84  
     checkAllgood, 85  
     collectAndPrint, 85  
     collectData, 86  
     HydrometerSensor, 84  
     operator<<, 89  
     printData, 87  
     stringStatus, 88  
 HydrometerSensor.cpp  
     operator<<, 321  
 id\_  
     Camera, 37  
     Sensor, 225  
 idAlarm\_  
     MonitoringSystem, 157  
 idCamera\_  
     MonitoringSystem, 157  
 idSelectedAlarm\_  
     MonitoringSystem, 157  
 idSelectedCamera\_  
     MonitoringSystem, 158  
 initialScreen  
     MonitoringSystem, 146  
 initialWindow  
     ScreenHardware, 202  
 inputCorrect

MonitoringSystem, 147  
isActive  
    Camera, 34  
    Hardware, 76  
    Sensor, 217  
isValidPrivileges  
    UsersDatabase, 291  
  
KEYBOARD  
    Hardware, 74  
keyboard  
    MonitoringSystem, 158  
KeyboardHardware, 90  
    ~KeyboardHardware, 92  
    askInput, 93  
    data\_, 95  
    displayOutput, 93  
    KeyboardHardware, 92  
    stringData\_, 95  
    stringInput, 94  
  
LIGHT\_SENSOR  
    Sensor, 211  
LightSensor, 95  
    ~LightSensor, 98  
    checkAllgood, 99  
    collectAndPrint, 99  
    collectData, 100  
    LightSensor, 98  
    operator<<, 103  
    printData, 101  
    stringStatus, 102  
LightSensor.cpp  
    operator<<, 325  
loadCameras  
    ManageCameras, 109  
loadSensorsData  
    AlarmSensors, 24  
loadSensorsDataBin  
    AlarmSensors, 24  
loadSensorsDataTxt  
    AlarmSensors, 25  
loadUsersFromFile  
    UsersServer, 301  
loginScreen  
    MonitoringSystem, 147  
loginWindow  
    ScreenHardware, 202  
  
main  
    main.cpp, 327  
    mainAirQualitySensor.cpp, 328  
    mainAlarmSensors.cpp, 329  
    mainGreenHouse.cpp, 331  
    mainHardware.cpp, 333  
    mainHydrometerSensor.cpp, 334  
    mainKeyboardHardware.cpp, 336  
    mainLightSensor.cpp, 338  
    mainManageCameras.cpp, 339  
    mainMonitoringSystem.cpp, 341  
    mainPhSensor.cpp, 343  
    mainPressureSensor.cpp, 344  
    mainRGBCamera.cpp, 345  
    mainScreenHardware.cpp, 347  
    mainSensor.cpp, 349  
    mainSwitchHardware.cpp, 352  
    mainTemperatureCamera.cpp, 353  
    mainTemperatureSensor.cpp, 355  
    mainUser.cpp, 356  
    mainUserAdmin.cpp, 359  
    mainUserEmployee.cpp, 361  
    mainUserGuest.cpp, 362  
    mainUsersDatabase.cpp, 364  
    mainUsersServer.cpp, 366  
    main.cpp  
        main, 327  
    MAIN\_MENU\_OPTIONS  
        MonitoringSystem.cpp, 371  
        MonitoringSystem.h, 373  
    mainAirQualitySensor.cpp  
        main, 328  
    mainAlarmSensors.cpp  
        main, 329  
    mainGreenHouse.cpp  
        main, 331  
    mainHardware.cpp  
        main, 333  
    mainHydrometerSensor.cpp  
        main, 334  
    mainKeyboardHardware.cpp  
        main, 336  
    mainLightSensor.cpp  
        main, 338  
    mainManageCameras.cpp  
        main, 339  
    mainMonitoringSystem.cpp  
        main, 341  
    mainPhSensor.cpp  
        main, 343  
    mainPressureSensor.cpp  
        main, 344  
    mainRGBCamera.cpp  
        main, 345  
    mainScreenHardware.cpp  
        main, 347  
    mainSensor.cpp  
        main, 349  
    mainSwitchHardware.cpp  
        main, 352  
    mainSystem  
        GreenHouse, 53  
    mainTemperatureCamera.cpp  
        main, 353  
    mainTemperatureSensor.cpp  
        main, 355  
    mainUser.cpp  
        main, 356

mainUserAdmin.cpp  
    main, 359  
mainUserEmployee.cpp  
    main, 361  
mainUserGuest.cpp  
    main, 362  
mainUsersDatabase.cpp  
    main, 364  
mainUsersServer.cpp  
    main, 366  
mainWindowAdmin  
    MonitoringSystem, 148  
    ScreenHardware, 203  
mainWindowEmployee  
    MonitoringSystem, 149  
    ScreenHardware, 204  
mainWindowGuest  
    MonitoringSystem, 150  
    ScreenHardware, 205  
manageAdmin  
    GreenHouse, 55  
ManageCameras, 103  
    ~ManageCameras, 105  
    addCamera, 105  
    cameras\_, 113  
    clearCameras, 106  
    collectAndDisplayData, 106  
    collectData, 106  
    createCamera, 107  
    displayAllCameras, 108  
    filename\_, 114  
    findCamera, 108  
    loadCameras, 109  
    ManageCameras, 105  
    removeCamera, 110  
    saveCameras, 111  
    turnOffCameras, 112  
    turnOnCameras, 113  
    turnOnOffCameras, 113  
manageCreateUser  
    GreenHouse, 59  
manageDeleteUser  
    GreenHouse, 59  
manageEmployee  
    GreenHouse, 60  
manageGuest  
    GreenHouse, 63  
manageLogin  
    GreenHouse, 64  
manageUpdateUser  
    GreenHouse, 67  
mc\_  
    GreenHouse, 70  
MonitoringSystem, 114  
    ~MonitoringSystem, 119  
    askEmail, 119  
    askIdAlarm, 120  
    askIdCamera, 121  
askInputInt, 121  
askMainUser, 122  
askName, 123  
askNIF, 124  
askPassword, 125  
askPrivileges, 126  
askTypeAlarm, 126  
askTypeCamera, 127  
cleanScreen, 128  
createAlarmScreen, 129  
createCameraScreen, 130  
createUserScreen, 130  
deleteAlarmScreen, 131  
deleteCameraScreen, 132  
deleteUserScreen, 133  
displayAlarmsScreen, 133  
displayCameraScreen, 134  
displayErrorScreen, 135  
displaySensorsScreen, 136  
displayUsersScreen, 137  
emailSelectedUser\_, 157  
exitScreen, 137  
getEmailSelectedUser, 138  
getIdAlarm, 139  
getIdCamera, 139  
getIdSelectedAlarm, 140  
getIdSelectedCamera, 140  
getName, 141  
getNameSelectedUser, 141  
getNIF, 142  
getNIFSelectedUser, 142  
getPassword, 143  
getPasswordSelectedUser, 143  
getPrivilegesSelectedUser, 144  
getSelection, 144  
getTypeAlarm, 145  
getTypeCamera, 145  
idAlarm\_, 157  
idCamera\_, 157  
idSelectedAlarm\_, 157  
idSelectedCamera\_, 158  
initialScreen, 146  
inputCorrect, 147  
keyboard, 158  
loginScreen, 147  
mainWindowAdmin, 148  
mainWindowEmployee, 149  
mainWindowGuest, 150  
MonitoringSystem, 119  
name\_, 158  
nameSelectedUser\_, 158  
nif\_, 159  
nifSelectedUser\_, 159  
password\_, 159  
passwordSelectedUser\_, 159  
privilegesSelectedUser\_, 160  
saveAlarmScreen, 151  
saveCameraScreen, 151

saveUsersScreen, 152  
screen, 160  
selection\_, 160  
selectUser, 153  
shortSelectUser, 154  
sw, 160  
turnOnOffCameraScreen, 154  
turnOnOffSystemScreen, 155  
typeAlarm\_, 161  
typeCamera\_, 161  
updateUserScreen, 156  
MonitoringSystem.cpp  
    ADMIN\_MENU\_OPTIONS, 370  
    EMPLOYEE\_MENU\_OPTIONS, 370  
    GUEST\_MENU\_OPTIONS, 370  
    MAIN\_MENU\_OPTIONS, 371  
MonitoringSystem.h  
    ADMIN\_MENU\_OPTIONS, 372  
    EMPLOYEE\_MENU\_OPTIONS, 372  
    GUEST\_MENU\_OPTIONS, 372  
    MAIN\_MENU\_OPTIONS, 373  
ms\_  
    GreenHouse, 71  
  
name  
    User, 263  
name\_  
    MonitoringSystem, 158  
nameSelectedUser\_  
    MonitoringSystem, 158  
nif  
    User, 263  
nif\_  
    MonitoringSystem, 159  
nifSelectedUser\_  
    MonitoringSystem, 159  
NONE  
    Hardware, 74  
    Sensor, 211  
  
operator<  
    Sensor, 217  
    User, 255  
operator<<  
    AirQualitySensor, 17  
    AirQualitySensor.cpp, 311  
    HydrometerSensor, 89  
    HydrometerSensor.cpp, 321  
    LightSensor, 103  
    LightSensor.cpp, 325  
    PhSensor, 170  
    PhSensor.cpp, 374  
    PressureSensor, 178  
    PressureSensor.cpp, 376  
    Sensor, 224  
    Sensor.cpp, 381  
    TemperatureSensor, 247  
    TemperatureSensor.cpp, 388  
    User, 262  
        User.cpp, 390  
operator>  
    Sensor, 218  
    User, 256  
operator>>  
    Sensor, 224  
    Sensor.cpp, 383  
    User, 262  
    User.cpp, 391  
operator()  
    UserNameComparator, 277  
    UserPtrComparator, 278  
operator==  
    Sensor, 218  
    User, 256  
  
password  
    User, 264  
password\_  
    MonitoringSystem, 159  
passwordSelectedUser\_  
    MonitoringSystem, 159  
PH\_SENSOR  
    Sensor, 211  
PhSensor, 162  
    ~PhSensor, 164  
    checkAllgood, 165  
    collectAndPrint, 165  
    collectData, 166  
    operator<<, 170  
    PhSensor, 164  
    printData, 167  
    qualifyPh, 168  
    stringStatus, 169  
PhSensor.cpp  
    operator<<, 374  
PRESSURE  
    Sensor, 211  
PressureSensor, 170  
    ~PressureSensor, 173  
    checkAllgood, 174  
    collectAndPrint, 174  
    collectData, 175  
    operator<<, 178  
    PressureSensor, 173  
    printData, 176  
    stringStatus, 177  
PressureSensor.cpp  
    operator<<, 376  
printCamera  
    Camera, 34  
    RGBCamera, 184  
    TemperatureCamera, 236  
printData  
    AirQualitySensor, 15  
    HydrometerSensor, 87  
    LightSensor, 101  
    PhSensor, 167  
    PressureSensor, 176

Sensor, 219  
 TemperatureSensor, 245  
 printDeletedUser  
     UsersDatabase, 292  
 printUser  
     User, 257  
 printUsers  
     UsersDatabase, 293  
 printUsersServer  
     UsersServer, 302  
 privileges  
     User, 264  
 privilegesSelectedUser\_  
     MonitoringSystem, 160  
  
 qualifyPh  
     PhSensor, 168  
  
 randomComponent  
     RGBCamera, 185  
 randomTemperature  
     TemperatureCamera, 237  
 readUsersFromFileBi  
     UsersServer, 302  
 readUsersFromFileTxt  
     UsersServer, 304  
 red\_  
     RGBCamera, 189  
 removeCamera  
     ManageCameras, 110  
 RGBCamera, 179  
     ~RGBCamera, 182  
     blue\_, 189  
     collectData, 182  
     getBlue, 183  
     getGreen, 183  
     getRed, 184  
     green\_, 189  
     printCamera, 184  
     randomComponent, 185  
     red\_, 189  
     RGBCamera, 181  
     setBlue, 186  
     setGreen, 187  
     setRed, 187  
     setRGB, 188  
  
 save  
     GreenHouse, 68  
 saveAlarmScreen  
     MonitoringSystem, 151  
 saveAlarmWindow  
     ScreenHardware, 205  
 saveCameras  
     ManageCameras, 111  
 saveCameraScreen  
     MonitoringSystem, 151  
 saveCameraWindow  
     ScreenHardware, 205  
  
 saveSensorsData  
     AlarmSensors, 25  
 saveSensorsDataBin  
     AlarmSensors, 26  
 saveSensorsDataTxt  
     AlarmSensors, 26  
 saveUsersScreen  
     MonitoringSystem, 152  
 saveUsersToFile  
     UsersServer, 305  
 saveUsersWindow  
     ScreenHardware, 206  
 SCREEN  
     Hardware, 74  
 screen  
     MonitoringSystem, 160  
 ScreenHardware, 190  
     ~ScreenHardware, 194  
     askInput, 194  
     cleanScreen, 195  
     createAlarmWindow, 196  
     createCameraWindow, 196  
     createUserWindow, 197  
     deleteAlarmWindow, 197  
     deleteCameraWindow, 198  
     deleteUserWindow, 198  
     displayAlarmsWindow, 199  
     displayCameraWindow, 199  
     displayErrorWindow, 200  
     displayOutput, 200  
     displaySensorsWindow, 201  
     displayUsersWindow, 201  
     exitWindow, 201  
     initialWindow, 202  
     loginWindow, 202  
     mainWindowAdmin, 203  
     mainWindowEmployee, 204  
     mainWindowGuest, 205  
     saveAlarmWindow, 205  
     saveCameraWindow, 205  
     saveUsersWindow, 206  
     ScreenHardware, 194  
     turnOnOffCameraWindow, 206  
     turnOnOffSystemWindow, 207  
     updateUserWindow, 207  
 ScreenHardware.cpp  
     ASK\_DATA, 379  
     USER\_PROMPT, 380  
 searchUser  
     GreenHouse, 68  
 selection\_  
     MonitoringSystem, 160  
 selectUser  
     MonitoringSystem, 153  
 Sensor, 208  
     ~Sensor, 212  
     active\_, 225  
     AIR\_QUALITY, 211

checkAllgood, 212  
collectAndPrint, 213  
collectData, 213  
data\_, 225  
getData, 214  
getId, 215  
getType, 216  
HYDROMETER, 211  
id\_, 225  
isActive, 217  
LIGHT\_SENSOR, 211  
NONE, 211  
operator<, 217  
operator<<, 224  
operator>, 218  
operator>>, 224  
operator==, 218  
PH\_SENSOR, 211  
PRESSURE, 211  
printData, 219  
Sensor, 211, 212  
setData, 219  
setId, 220  
setType, 221  
stringToType, 221  
TEMPERATURE, 211  
turnOff, 222  
turnOn, 222  
type\_, 225  
Types, 211  
typeToString, 223  
Sensor.cpp  
    operator<<, 381  
    operator>>, 383  
sensorExists  
    AlarmSensors, 26  
sensors\_  
    AlarmSensors, 29  
sensorsInitialized  
    AlarmSensors, 27  
setBlue  
    RGBCamera, 186  
setData  
    Sensor, 219  
setEmail  
    User, 257  
setGreen  
    RGBCamera, 187  
setId  
    Camera, 34  
    Sensor, 220  
setName  
    User, 258  
setNif  
    User, 259  
setPassword  
    User, 260  
setPrivileges  
        User, 261  
        setRed  
            RGBCamera, 187  
        setRGB  
            RGBCamera, 188  
        setTemperature  
            TemperatureCamera, 238  
        setType  
            Camera, 36  
            Hardware, 77  
            Sensor, 221  
        setUsers  
            UsersDatabase, 293  
        shortSelectUser  
            MonitoringSystem, 154  
        sleep  
            GreenHouse, 69  
        src/AirQualitySensor.cpp, 311  
        src/AirQualitySensor.h, 312  
        src/AlarmSensors.cpp, 313  
        src/AlarmSensors.h, 313  
        src/Camera.cpp, 315  
        src/Camera.h, 315  
        src/Exceptions.h, 316  
        src/GreenHouse.cpp, 317  
        src/GreenHouse.h, 318  
        src/Hardware.cpp, 319  
        src/Hardware.h, 320  
        src/HydrometerSensor.cpp, 321  
        src/HydrometerSensor.h, 322  
        src/KeyboardHardware.cpp, 323  
        src/KeyboardHardware.h, 323  
        src/LightSensor.cpp, 324  
        src/LightSensor.h, 325  
        src/main.cpp, 326  
        src/mainAirQualitySensor.cpp, 328  
        src/mainAlarmSensors.cpp, 329  
        src/mainGreenHouse.cpp, 330  
        src/mainHardware.cpp, 332  
        src/mainHydrometerSensor.cpp, 334  
        src/mainKeyboardHardware.cpp, 335  
        src/mainLightSensor.cpp, 337  
        src/mainManageCameras.cpp, 338  
        src/mainMonitoringSystem.cpp, 340  
        src/mainPhSensor.cpp, 342  
        src/mainPressureSensor.cpp, 344  
        src/mainRGBCamera.cpp, 345  
        src/mainScreenHardware.cpp, 346  
        src/mainSensor.cpp, 348  
        src/mainSwitchHardware.cpp, 351  
        src/mainTemperatureCamera.cpp, 353  
        src/mainTemperatureSensor.cpp, 354  
        src/mainUser.cpp, 356  
        src/mainUserAdmin.cpp, 358  
        src/mainUserEmployee.cpp, 360  
        src/mainUserGuest.cpp, 362  
        src/mainUsersDatabase.cpp, 363  
        src/mainUsersServer.cpp, 365

src/ManageCameras.cpp, 367  
 src/ManageCameras.h, 368  
 src/MonitoringSystem.cpp, 370  
 src/MonitoringSystem.h, 371  
 src/PhSensor.cpp, 373  
 src/PhSensor.h, 374  
 src/PressureSensor.cpp, 375  
 src/PressureSensor.h, 376  
 src/RGBCamera.cpp, 377  
 src/RGBCamera.h, 378  
 src/ScreenHardware.cpp, 379  
 src/ScreenHardware.h, 380  
 src/Sensor.cpp, 381  
 src/Sensor.h, 383  
 src/SwitchHardware.cpp, 384  
 src/SwitchHardware.h, 385  
 src/TemperatureCamera.cpp, 386  
 src/TemperatureCamera.h, 387  
 src/TemperatureSensor.cpp, 388  
 src/TemperatureSensor.h, 389  
 src/User.cpp, 390  
 src/User.h, 391  
 src/UserAdmin.cpp, 392  
 src/UserAdmin.h, 393  
 src/UserEmployee.cpp, 394  
 src/UserEmployee.h, 394  
 src/UserGuest.cpp, 395  
 src/UserGuest.h, 396  
 src/UsersDatabase.cpp, 397  
 src/UsersDatabase.h, 397  
 src/UsersServer.cpp, 398  
 src/UsersServer.h, 399  
 startSystem  
     GreenHouse, 69  
 status\_  
     AlarmSensors, 29  
 stringData\_  
     KeyboardHardware, 95  
 stringInput  
     KeyboardHardware, 94  
 stringStatus  
     AirQualitySensor, 16  
     Hardware, 78  
     HydrometerSensor, 88  
     LightSensor, 102  
     PhSensor, 169  
     PressureSensor, 177  
     TemperatureSensor, 246  
 stringToType  
     Hardware, 79  
     Sensor, 221  
 sw  
     MonitoringSystem, 160  
 SWITCH  
     Hardware, 74  
 SwitchHardware, 226  
     ~SwitchHardware, 228  
     askInput, 228  
 type\_

displayOutput, 229  
 SwitchHardware, 228  
 translateInput, 230

**TEMPERATURE**  
 Sensor, 211  
 temperature\_  
     TemperatureCamera, 239  
 TemperatureCamera, 231  
     ~TemperatureCamera, 235  
     collectData, 235  
     getTemperature, 236  
     printCamera, 236  
     randomTemperature, 237  
     setTemperature, 238  
     temperature\_, 239  
     TemperatureCamera, 234

TemperatureSensor, 239  
     ~TemperatureSensor, 242  
     checkAllgood, 243  
     collectAndPrint, 243  
     collectData, 244  
     operator<<, 247  
     printData, 245  
     stringStatus, 246  
     TemperatureSensor, 242

TemperatureSensor.cpp  
     operator<<, 388

translateInput  
     SwitchHardware, 230

turnOff  
     Camera, 36  
     Hardware, 80  
     Sensor, 222

turnOffCameras  
     ManageCameras, 112

turnOffSystem  
     AlarmSensors, 27

turnOn  
     Camera, 36  
     Hardware, 80  
     Sensor, 222

turnOnCameras  
     ManageCameras, 113

turnOnOffCameras  
     ManageCameras, 113

turnOnOffCameraScreen  
     MonitoringSystem, 154

turnOnOffCameraWindow  
     ScreenHardware, 206

turnOnOffSystem  
     AlarmSensors, 28

turnOnOffSystemScreen  
     MonitoringSystem, 155

turnOnOffSystemWindow  
     ScreenHardware, 207

turnOnSystem  
     AlarmSensors, 28

Camera, 37  
    Hardware, 81  
    Sensor, 225  
    typeAlarm\_  
        MonitoringSystem, 161  
    typeCamera\_  
        MonitoringSystem, 161  
    Types  
        Sensor, 211  
    Types\_Hardware  
        Hardware, 74  
    typeToString  
        Hardware, 80  
        Sensor, 223  
  
updateUser  
    UsersServer, 306  
updateUserScreen  
    MonitoringSystem, 156  
updateUserWindow  
    ScreenHardware, 207  
us\_  
    GreenHouse, 71  
User, 248  
    ~User, 252  
    email, 263  
    getEmail, 252  
    getName, 253  
    getNif, 254  
    getPassword, 254  
    getPrivileges, 255  
    name, 263  
    nif, 263  
    operator<, 255  
    operator<<, 262  
    operator>, 256  
    operator>>, 262  
    operator==, 256  
    password, 264  
    printUser, 257  
    privileges, 264  
    setEmail, 257  
    setName, 258  
    setNif, 259  
    setPassword, 260  
    setPrivileges, 261  
    User, 250, 251  
User.cpp  
    operator<<, 390  
    operator>>, 391  
USER\_PROMPT  
    ScreenHardware.cpp, 380  
UserAdmin, 265  
    ~UserAdmin, 268  
    UserAdmin, 266, 267  
UserEmployee, 269  
    ~UserEmployee, 272  
    UserEmployee, 270, 271  
UserGuest, 273  
    ~UserGuest, 276  
    UserGuest, 274, 275  
    UserNameComparator, 277  
        operator(), 277  
    UserPtrComparator, 278  
        operator(), 278  
users\_  
    UsersDatabase, 294  
UsersDatabase, 279  
    ~UsersDatabase, 281  
    addUser, 281  
    deleteUser, 282  
    deleteUserByEmail, 283  
    deleteUserByName, 284  
    deleteUserByNif, 285  
    findUser, 286  
    findUserByEmail, 287  
    findUserByName, 288  
    findUserByNif, 289  
    findUserByPassword, 290  
    getUsers, 290  
    getUsersSortedByName, 291  
    isValidPrivileges, 291  
    printDeletedUser, 292  
    printUsers, 293  
    setUsers, 293  
    users\_, 294  
    UsersDatabase, 280  
usersDatabase\_  
    UsersServer, 309  
UsersServer, 295  
    ~UsersServer, 298  
    createUser, 298  
    deleteUser, 299  
    fileNameBi\_, 308  
    fileNameTxt\_, 308  
    findUserLogin, 299  
    getPrivileges, 300  
    loadUsersFromFile, 301  
    printUsersServer, 302  
    readUsersFromFileBi, 302  
    readUsersFromFileTxt, 304  
    saveUsersToFile, 305  
    updateUser, 306  
    usersDatabase\_, 309  
    UsersServer, 297  
    writeUserToFileBi, 307  
    writeUserToFileTxt, 308  
  
    writeUserToFileBi  
        UsersServer, 307  
    writeUserToFileTxt  
        UsersServer, 308