My Project

Generated by Doxygen 1.16.0

Chapter 1

Smartgreenhouse

This project is an attempt to create a self-regulating greenhouse that would provide optimal groving conditions for high-humidity plants.

1.1 Sensors

- Gathers temperature and humidity from a DHT11 sensor module
- Measures the hours of light recieved by a photoresistorselement

1.2 Controls

- Regulates a mister, created by an ultrasonic atomizer module (exact model unknown)
- Simulates a heating element (regular LED used since no heating element where available)
- Simulates an UV light (regular LED used since no UV lamps where available)

1.3 Communication

The ESP runs an small server. Regulates the set targets.

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

File Documentation

3.1 climateControl.cpp File Reference

Climate controller: applies control logic (heater + mister) from SharedState sensor values/targets.

```
#include <Arduino.h>
#include "climateControl.h"
#include "heater.h"
#include "mister.h"
```

Functions

- void climateControlBegin ()

  Initialize controller timers/state.
- void climateControlUpdate (SharedState &state)

  Update climate control (heater + mister) based on sensor readings and targets.

Variables

- static const float TEMP_HYST = 0.5
- static const float HUM_HYST = 3.0
- static const uint32_t MIST_MAX_ON_MS = 20 * 1000
- static const uint32_t MIST_MIN_OFF_MS = 10 * 1000
- static uint32_t mistOnSinceMs = 0
- static uint32_t mistOffSinceMs = 0

3.1.1 Detailed Description

Climate controller: applies control logic (heater + mister) from SharedState sensor values/targets.

This module implements:

- Temperature control using hysteresis (prevents rapid toggling).
- Humidity control using hysteresis + safety timers (max ON time + minimum OFF backoff).

Typical use:

- call climateControlBegin() once in setup()
- call climateControlUpdate(state) repeatedly in loop()

3.1.2 Function Documentation

3.1.2.1 climateControlBegin()

```
void climateControlBegin ()
```

Initialize controller timers/state.

Call once in setup(). This does not change any outputs by itself.

3.1.2.2 climateControlUpdate()

```
void climateControlUpdate (
            SharedState & state)
```

Update climate control (heater + mister) based on sensor readings and targets.

Inputs (read from `state`):

- state.hasDht, state.tempC, state.humidityPct
- state.targetTempC, state.targetHumidityPct
- state.heaterOn, state.misterOn (used for hysteresis decisions)

Outputs (written to `state` and hardware):

- Heater output via heaterSet(state, on) (also updates state.heaterOn)
- Mister output via misterState(on) and state.misterOn

Fail-safe:

- If state.hasDht is false, heater and mister are turned OFF.

Parameters

state	Shared state containing latest sensor values, setpoints, and actuator states.

3.1.3 Variable Documentation

3.1.3.1 HUM_HYST

```
const float HUM_HYST = 3.0  [static]
```

3.1.3.2 MIST_MAX_ON_MS

```
const uint32_t MIST_MAX_ON_MS = 20 * 1000  [static]
```

3.1.3.3 MIST_MIN_OFF_MS

```
const uint32_t MIST_MIN_OFF_MS = 10 * 1000  [static]
```

3.1.3.4 mistOffSinceMs

```
uint32_t mistOffSinceMs = 0  [static]
```

3.1.3.5 mistOnSinceMs

```
uint32_t mistOnSinceMs = 0  [static]
```

3.1.3.6 TEMP_HYST

```
const float TEMP_HYST = 0.5  [static]
```

3.2 dht11Sensor.cpp File Reference

Implementing the DHT11 sensor.

```
#include "dht11Sensor.h"
#include <DHT.h>
```

Macros

- #define DHTPIN D7

  Pin used to on the microcontroller for signal to the sensor.
- #define DHTTYPE DHT11

  Type of sensor module. Used to correctly read information using the pre-build library.

Functions

- static DHT dht (DHTPIN, DHTTYPE)

  Internal DHT driver instance (module-owned).
- void dht11Begin ()

  Initialize the DHT11 hardware/library.
- void dht11Read (SharedState &state)

  Read temperature + humidity and write results into SharedState.

3.2.1 Detailed Description

Implementing the DHT11 sensor.

The file handles the reading from the DHT11 sensor and updating the associated sharedState

3.2.2 Macro Definition Documentation

3.2.2.1 DHTPIN

```
#define DHTPIN D7
```

Pin used to on the microcontroller for signal to the sensor.

3.2.2.2 DHTTYPE

```
#define DHTTYPE DHT11
```

Type of sensor module. Used to correctly read information using the pre-build library.

3.2.3 Function Documentation

3.2.3.1 dht()

```
DHT dht (
            DHTPIN ,
            DHTTYPE )  [static]
```

Internal DHT driver instance (module-owned).

3.2.3.2 dht11Begin()

```
void dht11Begin ()
```

Initialize the DHT11 hardware/library.

Call once in setup().

3.2.3.3 dht11Read()

```
void dht11Read (
            SharedState & state)
```

Read temperature + humidity and write results into SharedState.

On success:

- state.tempC, state.humidityPct updated
- state.hasDht = true

On failure:

- tempC/humidityPct set to NAN

- state.hasDht = false

Parameters

state	SharedState to update.

3.3 heater.cpp File Reference

Heater output implementation.

```
#include <Arduino.h>
#include "heater.h"
```

Functions

- void heaterBegin ()
  Initialize the heater output pin.
- void heaterSet (SharedState &state, bool on)
  Turn the output ON or OFF.

Variables

- static const uint8_t HEATER_LED_PIN = D2

3.3.1 Detailed Description

Heater output implementation.

The code states which pin is reserved for the heaping Lamp, and declares whether or not the LED is on

3.3.2 Internal state varibles

- HEATER_LED_PIN: Hardware pin for LED/heater lamp

3.3.3 Function Documentation

3.3.3.1 heaterBegin()

```
void heaterBegin ()
```

Initialize the heater output pin.

Configures the heater control/indicator pin as OUTPUT and ensures the heater starts in a safe OFF state.

- pinMode(HEATER_LED_PIN, OUTPUT)
- digitalWrite(HEATER_LED_PIN, LOW)

Note

Call once in setup() before using heaterSet().

3.3.3.2 heaterSet()

```
void heaterSet (
            SharedState & state,
            bool on)
```

Turn the output ON or OFF.

Writes the heater pin (HEATER_LED_PIN) and updates the shared state so the rest of the program knows the current heater status.

Parameters

state	SharedState to update
on	True to turn the heater ON, false to turn it OFF

3.3.4 Variable Documentation

3.3.4.1 HEATER_LED_PIN

```
const uint8_t HEATER_LED_PIN = D2  [static]
```

3.4 LightSensor.cpp File Reference

Light sensor (LDR) logic + lamp + daily light counter.

```
#include "LightSensor.h"
```

Functions

- static float clampf (float x, float lo, float hi)

  Clamp a floating-point value to a closed interval.
- void lightInit ()

  Initialize the light module (lamp output + daily tracking timer).
- void lightUpdate (SharedState &state)

  Initialize the LED sturn on and off method, and the intialize the clock tracking time.

Variables

- static const uint8_t LDR_PIN = A0
- static const uint8_t LEDPIN = D3
- static int threshold = 950
- static const float TARGET_HOURS = 6.0f
- static unsigned long lastMs = 0
- static unsigned long dayStartMs = 0
- static float hoursToday = 0.0f

3.4.1 Detailed Description

Light sensor (LDR) logic + lamp + daily light counter.

The code main job is to read/sensor how much sun light the plant need for optimal growth, the code uses the photosensor to get data on how sun light the plant get then with at set threshold either turns on or off the lamp for the plant for light

-

3.4.2 Internal state variables

- LDR_PIN: ADC pin for the LDR divider (A0)
- LEDPIN: Output pin for the lamp (D3)
- threshold: Light threshold to turn on LED/Lamp
- TARGET_HOURS: Daily target light hours
- lastMs: Last update timestamp
- dayStartMs: Start of day timestamp
- hoursToday: Accumulated effective light hours today

3.4.3 Function Documentation

3.4.3.1 clampf()

```
float clampf (
            float x,
            float lo,
            float hi)  [static]
```

Clamp a floating-point value to a closed interval.

Returns:

- lo if x $<$ lo
- hi if x $>$ hi
- x otherwise

Parameters

x	Input value.
lo	Lower bound (inclusive).
hi	Upper bound (inclusive).

Returns

Clamped value in the range [lo, hi].

3.4.3.2 lightInit()

```
void lightInit ()
```

Initialize the light module (lamp output + daily tracking timer).

Configures the lamp output pin and resets the internal timing state used to accumulate "effective light" hours for the current day.

- Sets LEDPIN as OUTPUT and turns the lamp OFF.
- Initializes timing variables (lastMs, dayStartMs) to millis().
- Resets hoursToday to 0.0.

Note

Call this once from setup() before calling lightUpdate().

3.4.3.3 lightUpdate()

```
void lightUpdate (
            SharedState & state)
```

Initialize the LED sturn on and off method, and the intialize the clock tracking time.

the method for turning on the lamp, has 2 requirments:

- First requirment is the raw data from dht is has to be lower than threshold, which means if the light is lower than the threshold you want than the first requirment is met.
- Second requirment is before turning on the lamp, the code checks if how long the plants has gotten light today, if the hours required isn't reached first then the lamp turns on.

Parameters

state	Sharedstate to update and communicate with overall main code

3.4.4 Variable Documentation

3.4.4.1 dayStartMs

```
unsigned long dayStartMs = 0  [static]
```

3.4.4.2 hoursToday

```
float hoursToday = 0.0f  [static]
```

3.4.4.3 lastMs

```
unsigned long lastMs = 0  [static]
```

3.4.4.4 LDR_PIN

```
const uint8_t LDR_PIN = A0  [static]
```

3.4.4.5 LEDPIN

```
const uint8_t LEDPIN = D3  [static]
```

3.4.4.6 TARGET_HOURS

```
const float TARGET_HOURS = 6.0f  [static]
```

3.4.4.7 threshold

```
int threshold = 950  [static]
```

3.5 mister.cpp File Reference

Initalises and controls the mister.

```
#include "mister.h"
```

Macros

- #define MISTERPIN D1

Functions

- void misterInit ()

  Initializes the mister control pin and sets it to a safe OFF state.
- void misterState (bool on)

  Turns the mister on or off. Handles unknown states by turning the mister off.

Variables

- static const bool MISTER_ACTIVE_LOW = true

3.5.1 Detailed Description

Initalises and controls the mister.

3.5.2 Internal state varible

- MISTER_ACTIVE_LOW: True if the mister control is active-low (LOW = ON, HIGH = OFF). The

3.5.3 Macro Definition Documentation

3.5.3.1 MISTERPIN

```
#define MISTERPIN D1
```

3.5.4 Function Documentation

3.5.4.1 misterInit()

```
void misterInit ()
```

Initializes the mister control pin and sets it to a safe OFF state.

Configures MISTERPIN as OUTPUT and writes the inactive level, (OFF) depending on whether the hardware is active: low or active: high.

3.5.4.2 misterState()

```
void misterState (
            bool on)
```

Turns the mister on or off. Handles unknown states by turning the mister off.

Parameters

on	can be true or false, to turn on or off the mister. It is inverted.

3.5.5 Variable Documentation

3.5.5.1 MISTER_ACTIVE_LOW

```
const bool MISTER_ACTIVE_LOW = true  [static]
```

3.6 README.md File Reference

3.7 Smartgreenhouse.ino File Reference

Main application that wires together all sensor + actuator modules.

```
#include <Arduino.h>
#include "sharedState.h"
#include "dht11Sensor.h"
#include "Lightsensor.h"
#include "heater.h"
#include "mister.h"
#include "climateControl.h"
```

Functions

- void setup ()

  Arduino setup function (runs once at boot).
- void loop ()

  Arduino main loop function (runs repeatedly forever).

Variables

- SharedState state

  Global shared state instance.
- static const uint32_t PRINT_MS = 1000

  Print interval for Serial status output (milliseconds).

3.7.1 Detailed Description

Main application that wires together all sensor + actuator modules.

Overall flow:

- setup(): initialize hardware/modules + set default targets
- loop(): read sensors -> run controller -> actuate outputs -> (optional) print status

3.7.2 Function Documentation

3.7.2.1 loop()

```
void loop ()
```

Arduino main loop function (runs repeatedly forever).

Typical pipeline per iteration: 1) Read sensors into SharedState 2) Run controller to decide actuator states 3) Actuator modules apply outputs (often inside controller or module update) 4) Periodically print system state for debugging

3.7.2.2 setup()

```
void setup ()
```

Arduino setup function (runs once at boot).

<

<

<

<

<

<

<

< Desired temperature in ℃

< Desired humidity in %

3.7.3 Variable Documentation

3.7.3.1 PRINT_MS

```
const uint32_t PRINT_MS = 1000  [static]
```

Print interval for Serial status output (milliseconds).

Used to avoid spamming Serial every loop iteration (loop runs very fast).

3.7.3.2 state

```
SharedState state
```

Global shared state instance.

All modules read/write through this object so the program has a single source of truth:

- sensors write temp/humidity/light info
- controller writes desired actuator states (heaterOn/misterOn/lampOn)
- actuators use these states to drive hardware

Index