Wemos Bridge Server

commit-4fb2e3f

Generated by Doxygen 1.9.8

Chapter 1

Wemos Bridge Server

Chapter 2

Test List

Member TEST (I2CClientTests, setup_ValidPort)

I2CClientTests.setup_ValidPort

Member TEST (I2CClientTests, setup_InvalidPort_Negative)

I2CClientTests.setup_InvalidPort_Negative

Member TEST (I2CClientTests, setup_InvalidPort_Zero)

I2CClientTests.setup_InvalidPort_Zero

Member TEST (I2CClientTests, setup_InvalidPort_High)

I2CClientTests.setup_InvalidPort_High

Member TEST (WemosServerTest, Constructor_ValidPort)

WemosServerTest.Constructor_ValidPort

Member TEST (WemosServerTest, Constructor_InvalidPort_Negative)

WemosServerTest.Constructor_InvalidPort_Negative

Member TEST (WemosServerTest, Constructor_InvalidPort_Zero)

WemosServerTest.Constructor_InvalidPort_Zero

Member TEST (WemosServerTest, Constructor_InvalidPort_High)

WemosServerTest.Constructor_InvalidPort_High

Member TEST (WemosServerTest, Constructor_ValidHubIPAddress)

WemosServerTest.Constructor_ValidHubIPAddress

Member TEST (WemosServerTest, Constructor_InvalidHubIPAddress)

WemosServerTest.Constructor_InvalidHubIPAddress

Member TEST (WemosServerTest, Constructor_ValidHubPort)

WemosServerTest.Constructor_ValidHubPort

Member TEST (WemosServerTest, Constructor_InvalidHubPort_Negative)

WemosServerTest.Constructor_InvalidHubPort_Negative

Member TEST (WemosServerTest, Constructor_InvalidHubPort_High)

WemosServerTest.Constructor_InvalidHubPort_High

Member TEST (WemosServerTest, Constructor_InvalidHubPort_Zero)

WemosServerTest.Constructror_InvalidHubPort_Zero

Chapter 3

Topic Index

3.1 Topics

Here is a list of all topics with brief descriptions:

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

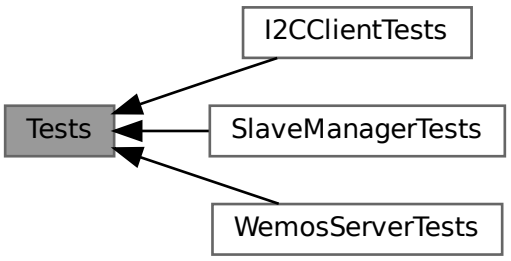
Chapter 6

Topic Documentation

6.1 Tests

Unit tests for the Wemos Bridge application.

Collaboration diagram for Tests:



Modules

- WemosServerTests

  All tests related to the WemosServer class.
- I2CClientTests

  All tests related to the I2CClient class.
- SlaveManagerTests

  All tests related to the SlaveManager class.

6.1.1 Detailed Description

Unit tests for the Wemos Bridge application.

6.1.2 WemosServerTests

All tests related to the WemosServer class.

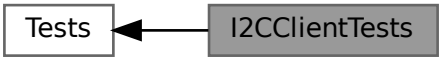
Collaboration diagram for WemosServerTests:

All tests related to the WemosServer class.

6.1.3 I2CClientTests

All tests related to the I2CClient class.

Collaboration diagram for I2CClientTests:



Functions

- TEST (I2CClientTests, setup_ValidPort)

  Test the setup() function with valid port numbers.
- TEST (I2CClientTests, setup_InvalidPort_Negative)
- TEST (I2CClientTests, setup_InvalidPort_Zero)
- TEST (I2CClientTests, setup_InvalidPort_High)

6.1.3.1 Detailed Description

All tests related to the I2CClient class.

6.1.3.2 Function Documentation

6.1.3.2.1 TEST() [1/4]

```
TEST (
            I2CClientTests ,
            setup_InvalidPort_High  )
```

Test I2CClientTests.setup_InvalidPort_High

- Verify that the setup() function throws an exception when a port number greater than 65535 is provided.
- Expects std::invalid_argument to be thrown.

Definition at line 57 of file test_i2cclient.cpp.

6.1.3.2.2 TEST() [2/4]

```
TEST (
            I2CClientTests ,
            setup_InvalidPort_Negative  )
```

Test I2CClientTests.setup_InvalidPort_Negative

- Verify that the setup() function throws an exception when a negative port number is provided.
- Expects std::invalid_argument to be thrown.

Definition at line 32 of file test_i2cclient.cpp.

6.1.3.2.3 TEST() [3/4]

```
TEST (
            I2CClientTests ,
            setup_InvalidPort_Zero  )
```

Test I2CClientTests.setup_InvalidPort_Zero

- Verify that the setup() function throws an exception when a port number of zero is provided.
- Expects std::invalid_argument to be thrown.

Definition at line 44 of file test_i2cclient.cpp.

6.1.3.2.4 TEST() [4/4]

```
TEST (
            I2CClientTests ,
            setup_ValidPort  )
```

Test the setup() function with valid port numbers.

Test I2CClientTests.setup_ValidPort

- Test the setup() function of I2CClient with valid port numbers.
- Expect no exceptions to be thrown.

Definition at line 18 of file test_i2cclient.cpp.

6.1.4 SlaveManagerTests

All tests related to the SlaveManager class.

Collaboration diagram for SlaveManagerTests:

All tests related to the SlaveManager class.

6.2 Packets

Contains all packet definitions in the application.

Classes

- struct sensor_header

  Header structure for sensor packets.
- struct sensor_metadata

  Structure for sensor metadata, which is always included in any packet.
- struct sensor_heartbeat

  Structure for heartbeat packets.
- struct sensor_packet_generic

  Structure for generic sensor packets.
- struct sensor_packet_temperature

  Structure for temperature sensor packets.
- struct sensor_packet_co2

  Structure for CO2 sensor packets.
- struct sensor_packet_humidity

  Structure for humidity sensor packets.
- struct sensor_packet_light

  Structure for light sensor packets.
- struct sensor_packet_rgb_light

  Structure for RGB light sensor packets.
- struct sensor_packet

  Union structure for the entire sensor packet.

6.2.1 Detailed Description

Contains all packet definitions in the application.

Warning

THESE MUST BE KEPT IN SYNC WITH OTHER SOFTWARE

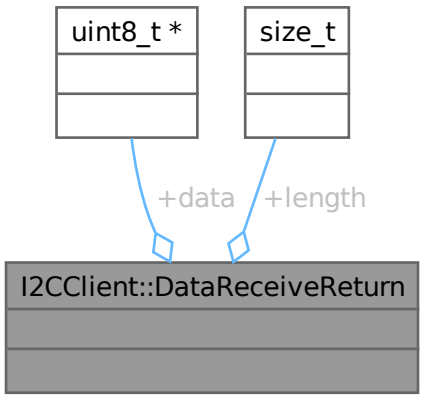
Chapter 7

Class Documentation

7.1 I2CClient::DataReceiveReturn Struct Reference

`#include <i2cclient.h>`

Collaboration diagram for I2CClient::DataReceiveReturn:



Public Attributes

- uint8_t * data
- size_t length

7.1.1 Detailed Description

Definition at line 52 of file i2cclient.h.

7.1.2 Member Data Documentation

7.1.2.1 data

```
uint8_t* I2CClient::DataReceiveReturn::data
```

Definition at line 53 of file i2cclient.h.

7.1.2.2 length

```
size_t I2CClient::DataReceiveReturn::length
```

Definition at line 54 of file i2cclient.h.

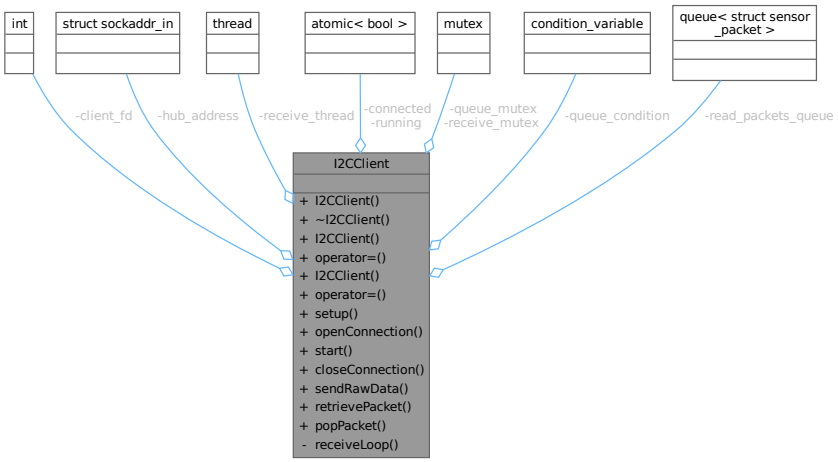
The documentation for this struct was generated from the following file:

- include/i2cclient.h

7.2 I2CClient Class Reference

```
#include <i2cclient.h>
```

Collaboration diagram for I2CClient:



Classes

- struct DataReceiveReturn

Public Member Functions

- I2CClient ()
  Constructor for I2CClient class.
- ~I2CClient ()
- I2CClient (const I2CClient &)=delete
- I2CClient & operator= (const I2CClient &)=delete
- I2CClient (I2CClient &&)=delete
- I2CClient & operator= (I2CClient &&)=delete
- void setup (const std::string &ip, int port)
  Initializes the settings necessary for connecting to the I2C hub.
- bool openConnection ()
  Connects to the I2C hub.
- void start ()
  Starts the I2C client.
- void closeConnection ()
  Disconnects from the I2C hub.
- void sendRawData (uint8_t *data, size_t length)
  Internal method to send data to the I2C hub.
- struct sensor_packet retrievePacket (bool block=false)
  Sends packet data to the I2C hub.
- struct sensor_packet popPacket ()

Private Member Functions

- void receiveLoop ()
  Internal receive loop for handling incoming data from the I2C hub.

Private Attributes

- int client_fd
- struct sockaddr_in hub_address
- std::thread receive_thread
- std::atomic< bool > connected
- std::atomic< bool > running
- std::mutex receive_mutex
- std::mutex queue_mutex
- std::condition_variable queue_condition
- std::queue< struct sensor_packet > read_packets_queue

7.2.1 Detailed Description

Definition at line 24 of file i2cclient.h.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 I2CClient() [1/3]

```
I2CClient::I2CClient ( )
```

Constructor for I2CClient class.

This constructor initializes the I2C client with the specified IP address and port.

Exceptions

std::invalid_argument	if the port number is invalid.

Warning

This constructor does not start the I2C client. Use setup(), openConnection() and start() instead.

Definition at line 28 of file i2cclient.cpp.

7.2.2.2 ~I2CClient()

```
I2CClient::~I2CClient ( )
```

Definition at line 32 of file i2cclient.cpp.

7.2.2.3 I2CClient() [2/3]

```
I2CClient::I2CClient (
            const I2CClient &  )  [delete]
```

7.2.2.4 I2CClient() [3/3]

```
I2CClient::I2CClient (
            I2CClient &&  )  [delete]
```

7.2.3 Member Function Documentation

7.2.3.1 closeConnection()

```
void I2CClient::closeConnection ( )
```

Disconnects from the I2C hub.

This method closes the connection to the I2C hub.

Definition at line 194 of file i2cclient.cpp.

7.2.3.2 openConnection()

```
bool I2CClient::openConnection ( )
```

Connects to the I2C hub.

This method establishes a connection to the I2C hub using the specified IP address and port.

Returns

true if the connection is successful, false otherwise.

Definition at line 150 of file i2cclient.cpp.

7.2.3.3 operator=() [1/2]

```
I2CClient & I2CClient::operator= (
            const I2CClient &  )  [delete]
```

7.2.3.4 operator=() [2/2]

```
I2CClient & I2CClient::operator= (
            I2CClient &&  )  [delete]
```

7.2.3.5 popPacket()

```
struct sensor_packet I2CClient::popPacket ( )
```

7.2.3.6 receiveLoop()

```
void I2CClient::receiveLoop ( )  [private]
```

Internal receive loop for handling incoming data from the I2C hub.

This method runs in a separate thread and continuously listens for incoming data from the I2C hub. It processes the received data and stores it in a buffer for later use.

Warning

This method should not be called directly. It is intended to be used internally by the class.

Definition at line 45 of file i2cclient.cpp.

7.2.3.7 retrievePacket()

```
struct sensor_packet I2CClient::retrievePacket (
            bool block = false )
```

Sends packet data to the I2C hub.

Parameters

t.↩ b.d.	

Exceptions

std::runtime_error	if sending data fails.

Receives data from the I2C hub.

Parameters

block	Whether or not to block until a packet can be retrieved

Returns

A struct containing the received packet data.

Exceptions

std::runtime_error	if receiving data fails.

Definition at line 213 of file i2cclient.cpp.

7.2.3.8 sendRawData()

```
void I2CClient::sendRawData (
            uint8_t * data,
            size_t length )
```

Internal method to send data to the I2C hub.

Parameters

data	The data to send to the I2C hub.
length	The length of the data to send.

Exceptions

std::runtime_error	if sending data fails.

Definition at line 206 of file i2cclient.cpp.

7.2.3.9 setup()

```
void I2CClient::setup (
            const std::string & ip,
            int port )
```

Initializes the settings necessary for connecting to the I2C hub.

This method initializes the remote address details (IP address and port) for the I2C hub to connect to.

Parameters

ip	The IP address of the I2C hub.
port	The port number of the I2C hub.

Exceptions

std::invalid_argument	if an invalid IP address or port is passed

Definition at line 138 of file i2cclient.cpp.

7.2.3.10 start()

```
void I2CClient::start ( )
```

Starts the I2C client.

This method starts the I2C client and begins listening for incoming data from the I2C hub.

Exceptions

std::runtime_error	if the client is not connected to the hub.

Definition at line 182 of file i2cclient.cpp.

7.2.4 Member Data Documentation

7.2.4.1 client_fd

```
int I2CClient::client_fd  [private]
```

Definition at line 26 of file i2cclient.h.

7.2.4.2 connected

```
std::atomic<bool> I2CClient::connected  [private]
```

Definition at line 32 of file i2cclient.h.

7.2.4.3 hub_address

```
struct sockaddr_in I2CClient::hub_address  [private]
```

Definition at line 28 of file i2cclient.h.

7.2.4.4 queue_condition

```
std::condition_variable I2CClient::queue_condition  [private]
```

Definition at line 38 of file i2cclient.h.

7.2.4.5 queue_mutex

```
std::mutex I2CClient::queue_mutex  [private]
```

Definition at line 36 of file i2cclient.h.

7.2.4.6 read_packets_queue

```
std::queue<struct sensor_packet> I2CClient::read_packets_queue  [private]
```

Definition at line 40 of file i2cclient.h.

7.2.4.7 receive_mutex

```
std::mutex I2CClient::receive_mutex  [private]
```

Definition at line 35 of file i2cclient.h.

7.2.4.8 receive_thread

```
std::thread I2CClient::receive_thread  [private]
```

Definition at line 30 of file i2cclient.h.

7.2.4.9 running

```
std::atomic<bool> I2CClient::running  [private]
```

Definition at line 33 of file i2cclient.h.

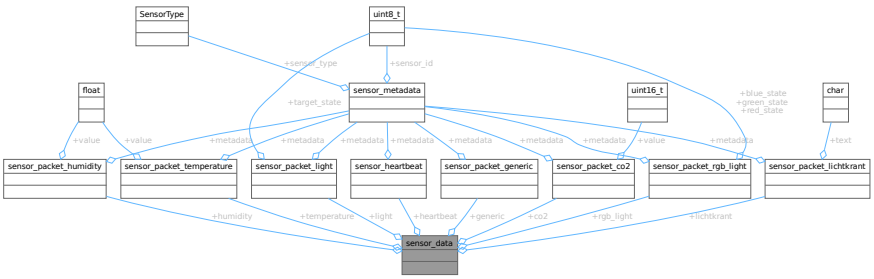
The documentation for this class was generated from the following files:

- include/i2cclient.h
- src/i2cclient.cpp

7.3 sensor_data Union Reference

```
#include <packets.h>
```

Collaboration diagram for sensor_data:



Public Attributes

- struct sensor_heartbeat heartbeat
- struct sensor_packet_generic generic
- struct sensor_packet_temperature temperature
- struct sensor_packet_co2 co2
- struct sensor_packet_humidity humidity
- struct sensor_packet_light light
- struct sensor_packet_rgb_light rgb_light
- struct sensor_packet_lichtkrant lichtkrant

7.3.1 Detailed Description

Definition at line 4 of file packets.h.

7.3.2 Member Data Documentation

7.3.2.1 co2

```
struct sensor_packet_co2 sensor_data::co2
```

Definition at line 8 of file packets.h.

7.3.2.2 generic

```
struct sensor_packet_generic sensor_data::generic
```

Definition at line 6 of file packets.h.

7.3.2.3 heartbeat

```
struct sensor_heartbeat sensor_data::heartbeat
```

Definition at line 5 of file packets.h.

7.3.2.4 humidity

```
struct sensor_packet_humidity sensor_data::humidity
```

Definition at line 9 of file packets.h.

7.3.2.5 lichtkrant

```
struct sensor_packet_lichtkrant sensor_data::lichtkrant
```

Definition at line 12 of file packets.h.

7.3.2.6 light

```
struct sensor_packet_light sensor_data::light
```

Definition at line 10 of file packets.h.

7.3.2.7 rgb_light

```
struct sensor_packet_rgb_light sensor_data::rgb_light
```

Definition at line 11 of file packets.h.

7.3.2.8 temperature

```
struct sensor_packet_temperature sensor_data::temperature
```

Definition at line 7 of file packets.h.

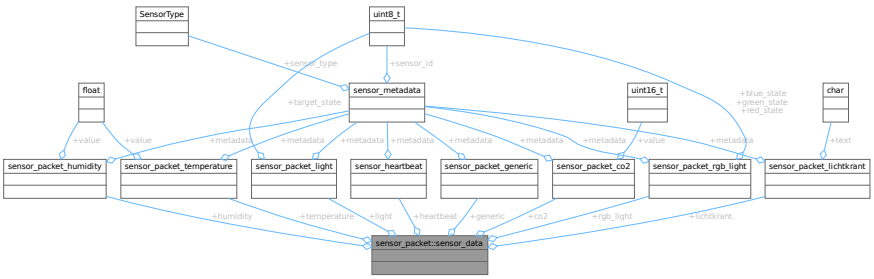
The documentation for this union was generated from the following file:

- include/packets.h

7.4 sensor_packet::sensor_data Union Reference

```
#include <packets.h>
```

Collaboration diagram for sensor_packet::sensor_data:



Public Attributes

- struct sensor_heartbeat heartbeat
- struct sensor_packet_generic generic
- struct sensor_packet_temperature temperature
- struct sensor_packet_co2 co2
- struct sensor_packet_humidity humidity
- struct sensor_packet_light light
- struct sensor_packet_rgb_light rgb_light
- struct sensor_packet_lichtkrant lichtkrant

7.4.1 Detailed Description

Definition at line 239 of file packets.h.

7.4.2 Member Data Documentation

7.4.2.1 co2

```
struct sensor_packet_co2 sensor_packet::sensor_data::co2
```

Definition at line 243 of file packets.h.

7.4.2.2 generic

```
struct sensor_packet_generic sensor_packet::sensor_data::generic
```

Definition at line 241 of file packets.h.

7.4.2.3 heartbeat

```
struct sensor_heartbeat sensor_packet::sensor_data::heartbeat
```

Definition at line 240 of file packets.h.

7.4.2.4 humidity

```
struct sensor_packet_humidity sensor_packet::sensor_data::humidity
```

Definition at line 244 of file packets.h.

7.4.2.5 lichtkrant

`struct sensor_packet_lichtkrant sensor_packet::sensor_data::lichtkrant`

Definition at line 247 of file packets.h.

7.4.2.6 light

`struct sensor_packet_light sensor_packet::sensor_data::light`

Definition at line 245 of file packets.h.

7.4.2.7 rgb_light

`struct sensor_packet_rgb_light sensor_packet::sensor_data::rgb_light`

Definition at line 246 of file packets.h.

7.4.2.8 temperature

`struct sensor_packet_temperature sensor_packet::sensor_data::temperature`

Definition at line 242 of file packets.h.

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

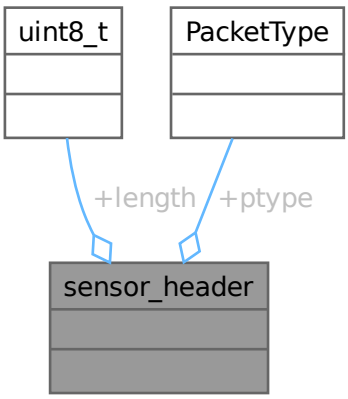
- include/packets.h

7.5 sensor_header Struct Reference

Header structure for sensor packets.

`#include <packets.h>`

Collaboration diagram for sensor_header:



Public Attributes

- uint8_t length

  Length of the packet excluding the header.
- PacketType ptype

  Type of the packet as PacketType (DATA, HEARTBEAT, etc.).

7.5.1 Detailed Description

Header structure for sensor packets.

Definition at line 41 of file packets.h.

7.5.2 Member Data Documentation

7.5.2.1 length

```
uint8_t sensor_header::length
```

Length of the packet excluding the header.

Definition at line 43 of file packets.h.

7.5.2.2 ptype

```
PacketType sensor_header::ptype
```

Type of the packet as PacketType (DATA, HEARTBEAT, etc.).

Definition at line 45 of file packets.h.

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

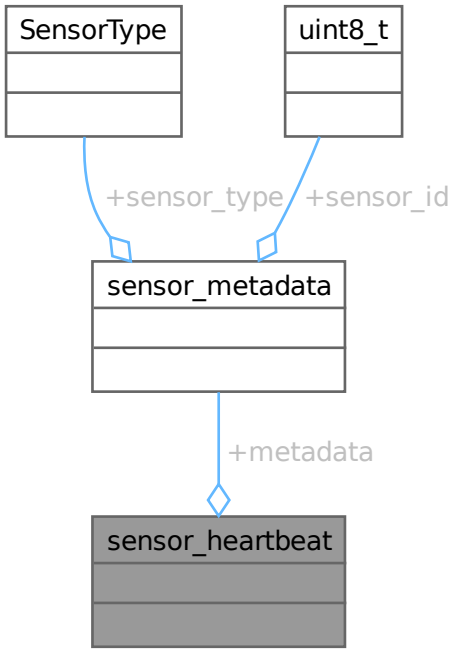
- include/packets.h

7.6 sensor_heartbeat Struct Reference

Structure for heartbeat packets.

`#include <packets.h>`

Collaboration diagram for sensor_heartbeat:



Public Attributes

- struct sensor_metadata metadata

7.6.1 Detailed Description

Structure for heartbeat packets.

This structure contains the type and ID of the sensor being addressed. This structure is used for heartbeat packets sent by the sensors to indicate they are still alive.

Definition at line 70 of file packets.h.

7.6.2 Member Data Documentation

7.6.2.1 metadata

```
struct sensor_metadata sensor_heartbeat::metadata
```

Definition at line 71 of file packets.h.

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

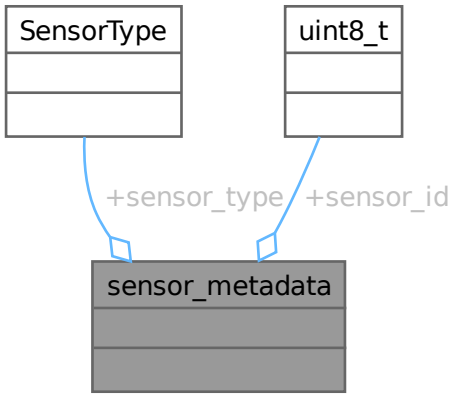
- include/packets.h

7.7 sensor_metadata Struct Reference

Structure for sensor metadata, which is always included in any packet.

```
#include <packets.h>
```

Collaboration diagram for sensor_metadata:



Public Attributes

- SensorType sensor_type

  Type of the sensor being addressed as SensorType (one byte)
- uint8_t sensor_id

  ID of the sensor being addressed.

7.7.1 Detailed Description

Structure for sensor metadata, which is always included in any packet.

Definition at line 53 of file packets.h.

7.7.2 Member Data Documentation

7.7.2.1 sensor_id

```
uint8_t sensor_metadata::sensor_id
```

ID of the sensor being addressed.

Definition at line 57 of file packets.h.

7.7.2.2 sensor_type

```
SensorType sensor_metadata::sensor_type
```

Type of the sensor being addressed as SensorType (one byte)

Definition at line 55 of file packets.h.

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

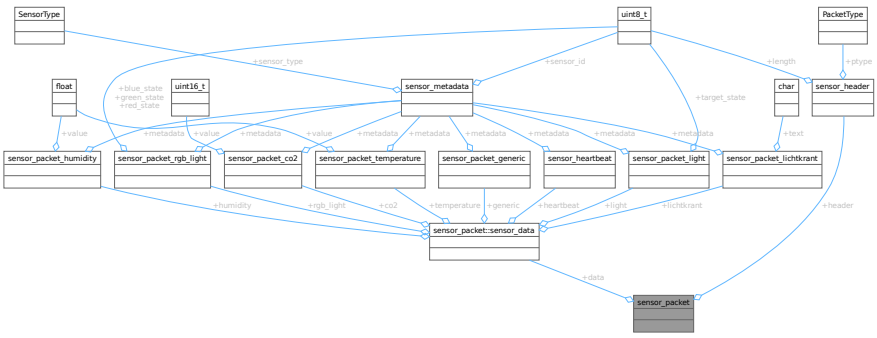
- include/packets.h

7.8 sensor_packet Struct Reference

Union structure for the entire sensor packet.

```
#include <packets.h>
```

Collaboration diagram for sensor_packet:



Classes

- union sensor_data

Public Attributes

- struct sensor_header header

  Header of the packet containing length and type information.
- union sensor_packet::sensor_data data

7.8.1 Detailed Description

Union structure for the entire sensor packet.

This structure is used to encapsulate the different types of sensor packets that can be sent and has the shape of a valid packet.

It contains a sensor_header followed by a union of different sensor data types. The union allows for different types of sensor data to be stored in the same memory location, depending on the packet type.

Example usage:
```
sensor_packet packet;
packet.header.length = sizeof(sensor_packet_generic);
packet.header.ptype = PacketType::DATA;
packet.data.generic.metadata.sensor_type = SensorType::BUTTON;
packet.data.generic.metadata.sensor_id = 1;

// Accessing the packet data
if (packet.header.ptype == PacketType::DATA) {
    if (packet.data.generic.metadata.sensor_type == SensorType::BUTTON) {
        uint8_t sensor_id = packet.data.generic.metadata.sensor_id;
        // Process button press event for sensor_id
    }
}
```

To use this structure to request data from the dashboard, you can set the ptype to DASHBOARD_GET to indicate that you want to request data from the backend (wemos bridge). Then, you use a sensor_packet_generic to specify the type of sensor you want to request data for and the ID of that sensor.

Example: We want to request temperature data from the backend (wemos bridge) for sensor ID 1.
```
sensor_packet packet;
packet.header.length = sizeof(sensor_packet_generic);
packet.header.ptype = PacketType::DASHBOARD_GET;
packet.data.generic.metadata.sensor_type = SensorType::TEMPERATURE;
packet.data.generic.metadata.sensor_id = 1;
```

The backend (wemos bridge) will then respond with a packet of type DASHBOARD_RESPONSE containing the requested data. Following the correct type packet for this example would be a sensor_packet_temperature.

Example: We want to change the color of an RGB light with ID 1 to red (255, 0, 0).
```
sensor_packet packet;
packet.header.length = sizeof(sensor_packet_rgb_light);
packet.header.ptype = PacketType::DASHBOARD_POST;
packet.data.rgb_light.metadata.sensor_type = SensorType::RGB_LIGHT;
packet.data.rgb_light.metadata.sensor_id = 1;
packet.data.rgb_light.red_state = 255;
packet.data.rgb_light.green_state = 0;
packet.data.rgb_light.blue_state = 0;
```

Note

The data field is a union that can hold different types of sensor data.

Definition at line 234 of file packets.h.

7.8.2 Member Data Documentation

7.8.2.1 data

```
union sensor_packet::sensor_data sensor_packet::data
```

7.8.2.2 header

```
struct sensor_header sensor_packet::header
```

Header of the packet containing length and type information.

Definition at line 236 of file packets.h.

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

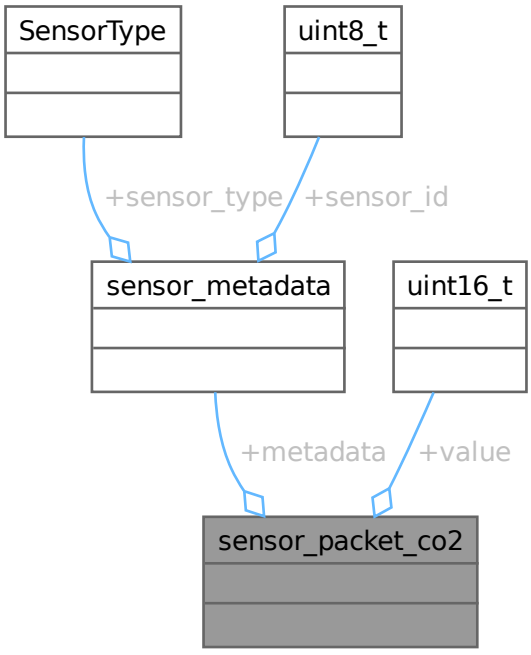
- include/packets.h

7.9 sensor_packet_co2 Struct Reference

Structure for CO2 sensor packets.

```
#include <packets.h>
```

Collaboration diagram for sensor_packet_co2:



Public Attributes

- struct sensor_metadata metadata
- uint16_t value

  Value of the sensor reading the CO2 level represented in ppm.

7.9.1 Detailed Description

Structure for CO2 sensor packets.

This structure contains the type, ID, and value of the CO2 sensor reading.

Note

The CO2 value is represented in parts per million (ppm).

Definition at line 108 of file packets.h.

7.9.2 Member Data Documentation

7.9.2.1 metadata

```
struct sensor_metadata sensor_packet_co2::metadata
```

Definition at line 109 of file packets.h.

7.9.2.2 value

```
uint16_t sensor_packet_co2::value
```

Value of the sensor reading the CO2 level represented in ppm.

Definition at line 111 of file packets.h.

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

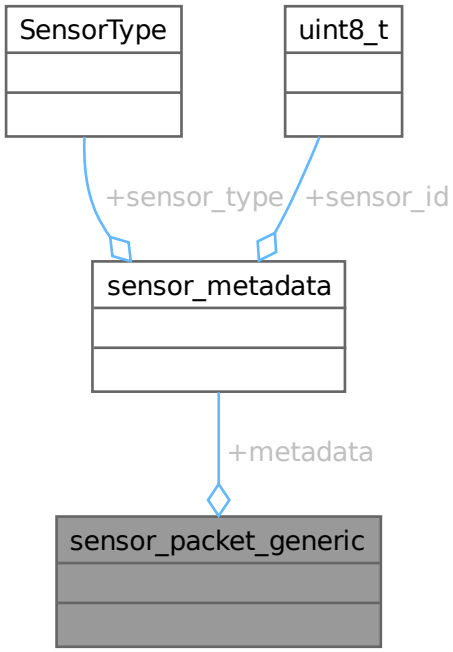
- include/packets.h

7.10 sensor_packet_generic Struct Reference

Structure for generic sensor packets.

`#include <packets.h>`

Collaboration diagram for sensor_packet_generic:



Public Attributes

- struct sensor_metadata metadata

7.10.1 Detailed Description

Structure for generic sensor packets.

This structure contains the type and ID of the sensor being addressed. This structure is used for generic sensor packets that do not require additional data. For example, it can be used for a simple button press event.

Definition at line 82 of file packets.h.

7.10.2 Member Data Documentation

7.10.2.1 metadata

```
struct sensor_metadata sensor_packet_generic::metadata
```

Definition at line 83 of file packets.h.

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

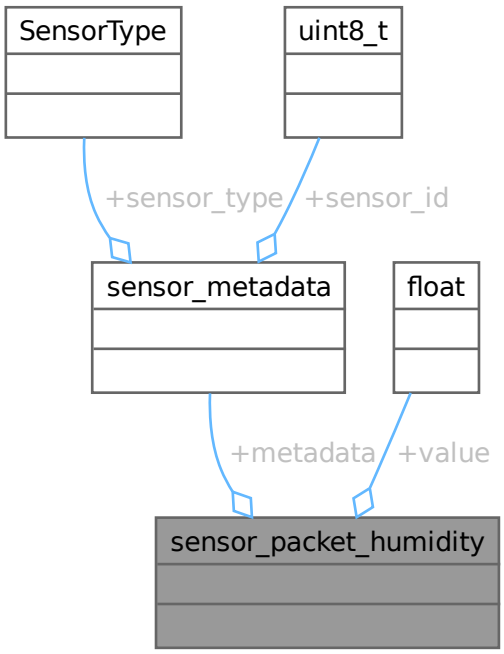
- include/packets.h

7.11 sensor_packet_humidity Struct Reference

Structure for humidity sensor packets.

```
#include <packets.h>
```

Collaboration diagram for sensor_packet_humidity:



Public Attributes

- struct sensor_metadata metadata
- float value

  Value of the sensor reading the humidity level represented in percentage.

7.11.1 Detailed Description

Structure for humidity sensor packets.

This structure contains the type, ID, and value of the humidity sensor reading.

Note

The humidity value is represented in percentage.

Definition at line 121 of file packets.h.

7.11.2 Member Data Documentation

7.11.2.1 metadata

```
struct sensor_metadata sensor_packet_humidity::metadata
```

Definition at line 122 of file packets.h.

7.11.2.2 value

```
float sensor_packet_humidity::value
```

Value of the sensor reading the humidity level represented in percentage.

Definition at line 124 of file packets.h.

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

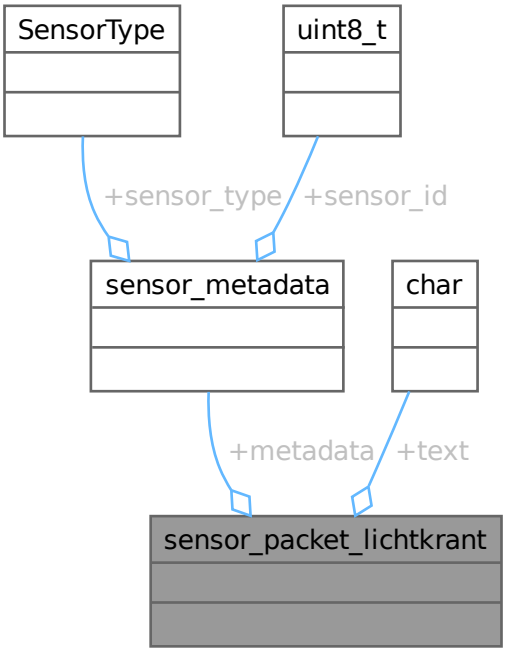
- include/packets.h

7.12 sensor_packet_lichtkrant Struct Reference

structure for the lichkrant packets

```
#include <packets.h>
```

Collaboration diagram for sensor_packet_lichtkrant:



Public Attributes

- struct sensor_metadata metadata
- char text [16]

7.12.1 Detailed Description

structure for the lichkrant packets

Contains the string to display on the lichtkrant @ingroupo Packets

Definition at line 164 of file packets.h.

7.12.2 Member Data Documentation

7.12.2.1 metadata

```
struct sensor_metadata sensor_packet_lichtkrant::metadata
```

Definition at line 165 of file packets.h.

7.12.2.2 text

```
char sensor_packet_lichtkrant::text[16]
```

Definition at line 166 of file packets.h.

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

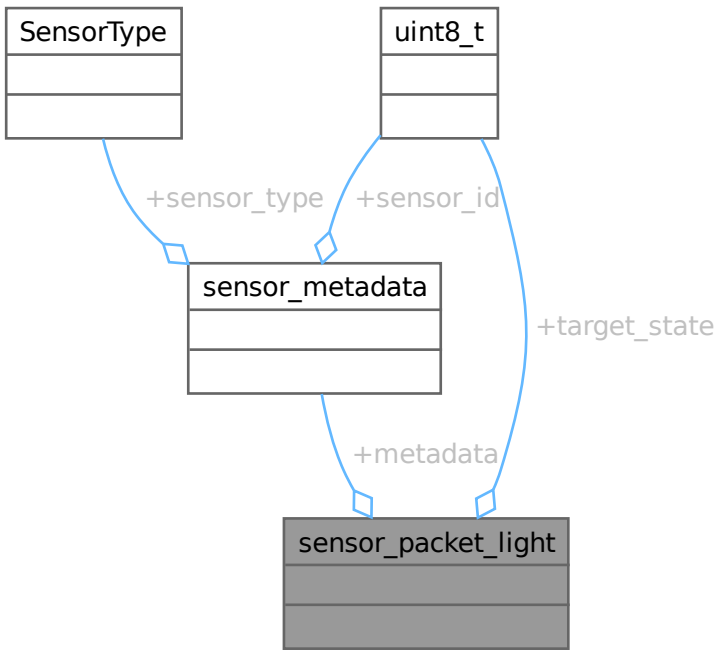
- include/packets.h

7.13 sensor_packet_light Struct Reference

Structure for light sensor packets.

```
#include <packets.h>
```

Collaboration diagram for sensor_packet_light:



Public Attributes

- struct sensor_metadata metadata
- uint8_t target_state

  Target state of the light (on 1/off 0) represented as a boolean value.

7.13.1 Detailed Description

Structure for light sensor packets.

This structure contains the type, ID, and target state of the light/led. This structure is used for light control packets sent to the light/led.

Definition at line 134 of file packets.h.

7.13.2 Member Data Documentation

7.13.2.1 metadata

```
struct sensor_metadata sensor_packet_light::metadata
```

Definition at line 135 of file packets.h.

7.13.2.2 target_state

```
uint8_t sensor_packet_light::target_state
```

Target state of the light (on 1/off 0) represented as a boolean value.

Definition at line 137 of file packets.h.

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

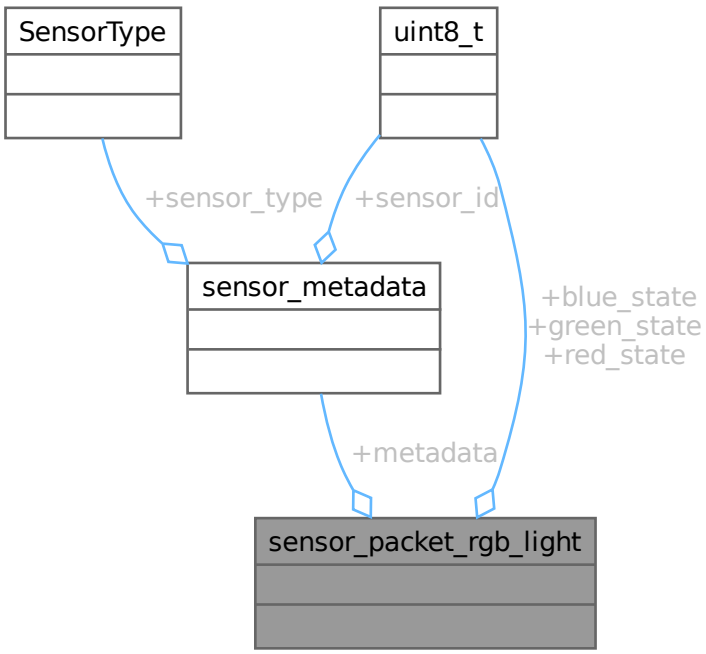
- include/packets.h

7.14 sensor_packet_rgb_light Struct Reference

Structure for RGB light sensor packets.

```
#include <packets.h>
```

Collaboration diagram for sensor_packet_rgb_light:



Public Attributes

- struct sensor_metadata metadata
- uint8_t red_state

  Target state of the red color (0-255) represented as an 8-bit integer.
- uint8_t green_state

  Target state of the green color (0-255) represented as an 8-bit integer.
- uint8_t blue_state

  Target state of the blue color (0-255) represented as an 8-bit integer.

7.14.1 Detailed Description

Structure for RGB light sensor packets.

This structure contains the type, ID, and target color of the RGB light. This structure is used for RGB light control packets sent to the RGB light.

Note

The RGB values are represented as 8-bit integers (0-255).

Definition at line 148 of file packets.h.

7.14.2 Member Data Documentation

7.14.2.1 blue_state

```
uint8_t sensor_packet_rgb_light::blue_state
```

Target state of the blue color (0-255) represented as an 8-bit integer.

Definition at line 155 of file packets.h.

7.14.2.2 green_state

```
uint8_t sensor_packet_rgb_light::green_state
```

Target state of the green color (0-255) represented as an 8-bit integer.

Definition at line 153 of file packets.h.

7.14.2.3 metadata

```
struct sensor_metadata sensor_packet_rgb_light::metadata
```

Definition at line 149 of file packets.h.

7.14.2.4 red_state

```
uint8_t sensor_packet_rgb_light::red_state
```

Target state of the red color (0-255) represented as an 8-bit integer.

Definition at line 151 of file packets.h.

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

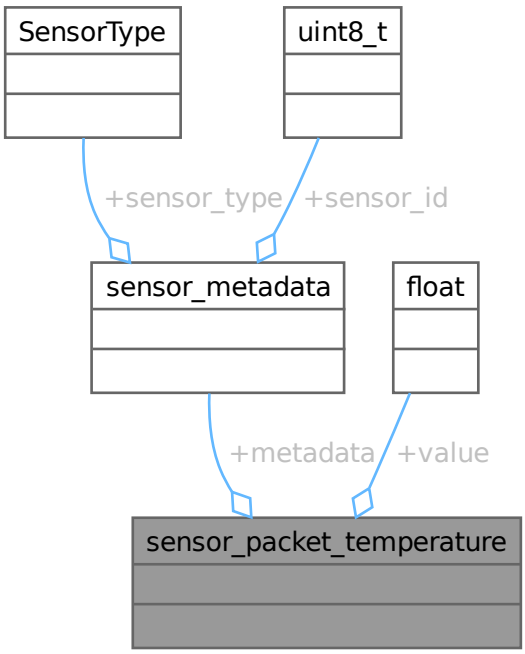
- include/packets.h

7.15 sensor_packet_temperature Struct Reference

Structure for temperature sensor packets.

`#include <packets.h>`

Collaboration diagram for sensor_packet_temperature:



Public Attributes

- struct sensor_metadata metadata
- float value

  Value of the sensor reading the temperature represented in Celcius.

7.15.1 Detailed Description

Structure for temperature sensor packets.

This structure contains the type, ID, and value of the temperature sensor reading.

Note

The temperature value is represented in Celsius.

Definition at line 95 of file packets.h.

7.15.2 Member Data Documentation

7.15.2.1 metadata

```
struct sensor_metadata sensor_packet_temperature::metadata
```

Definition at line 96 of file packets.h.

7.15.2.2 value

```
float sensor_packet_temperature::value
```

Value of the sensor reading the temperature represented in Celcius.

Definition at line 98 of file packets.h.

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

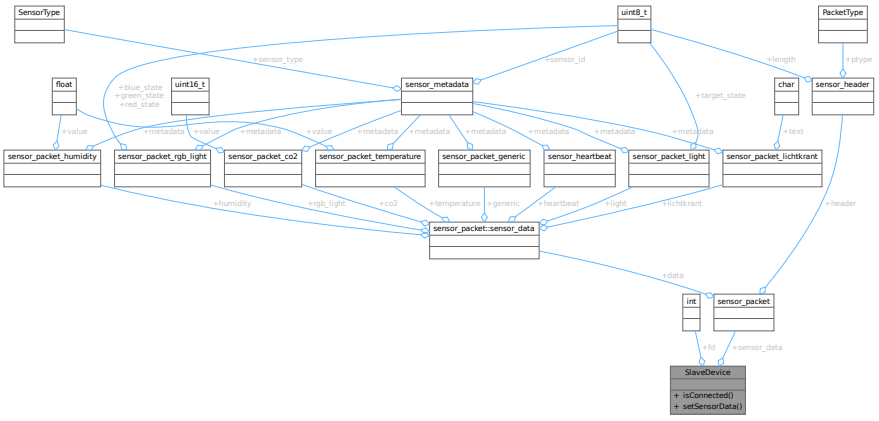
- include/packets.h

7.16 SlaveDevice Struct Reference

Structure representing a slave device.

```
#include <slavemanager.h>
```

Collaboration diagram for SlaveDevice:



Public Member Functions

- bool isConnected () const
- void setSensorData (const struct sensor_packet &)

Public Attributes

- int fd = -1
- struct sensor_packet sensor_data = {0}

7.16.1 Detailed Description

Structure representing a slave device.

This structure contains the file descriptor associated with the slave device, and also its current state in the form of a packet.

Definition at line 28 of file slavemanager.h.

7.16.2 Member Function Documentation

7.16.2.1 isConnected()

```
bool SlaveDevice::isConnected ( ) const
```

Definition at line 20 of file slavemanager.cpp.

7.16.2.2 setSensorData()

```
void SlaveDevice::setSensorData (
            const struct sensor_packet & pkt )
```

Definition at line 21 of file slavemanager.cpp.

7.16.3 Member Data Documentation

7.16.3.1 fd

```
int SlaveDevice::fd = -1
```

Definition at line 29 of file slavemanager.h.

7.16.3.2 sensor_data

```
struct sensor_packet SlaveDevice::sensor_data = {0}
```

Definition at line 30 of file slavemanager.h.

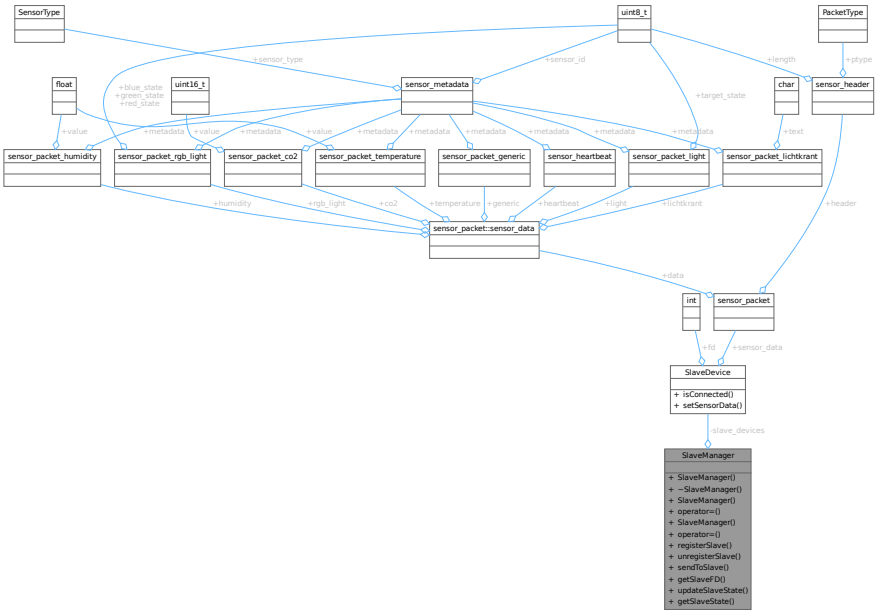
The documentation for this struct was generated from the following files:

- include/slavemanager.h
- src/slavemanager.cpp

7.17 SlaveManager Class Reference

`#include <slavemanager.h>`

Collaboration diagram for SlaveManager:



Public Member Functions

- SlaveManager ()
- ∼SlaveManager ()
- SlaveManager (const SlaveManager &)=delete
- SlaveManager & operator= (const SlaveManager &)=delete
- SlaveManager (SlaveManager &&)=delete
- SlaveManager & operator= (SlaveManager &&)=delete
- void registerSlave (uint8_t slave_id, int fd)

  Registers a slave device with the given ID and file descriptor.
- void unregisterSlave (uint8_t slave_id)

  Unregisters a slave device with the given ID.
- int sendToSlave (uint8_t slave_id, const void ∗data, size_t length)

  Sends data to the slave device with the given ID.
- int getSlaveFD (uint8_t slave_id) const

  Gets the file descriptor associated with the given slave ID.
- void updateSlaveState (uint8_t slave_id, const struct sensor_packet &packet)

  Updates the internal sensor_packet structure that contains the current state of the device.
- struct sensor_packet getSlaveState (uint8_t slave_id)

  Retrieves the internal sensor_packet structure that contains the current state of the device.

Private Attributes

- SlaveDevice slave_devices [MAX_SLAVE_ID+1]

7.17.1 Detailed Description

Definition at line 36 of file slavemanager.h.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 SlaveManager() [1/3]

```
SlaveManager::SlaveManager ( )
```

Definition at line 25 of file slavemanager.cpp.

7.17.2.2 ~SlaveManager()

```
SlaveManager::~SlaveManager ( )
```

Definition at line 31 of file slavemanager.cpp.

7.17.2.3 SlaveManager() [2/3]

```
SlaveManager::SlaveManager (
            const SlaveManager &  )  [delete]
```

7.17.2.4 SlaveManager() [3/3]

```
SlaveManager::SlaveManager (
            SlaveManager &&  )  [delete]
```

7.17.3 Member Function Documentation

7.17.3.1 getSlaveFD()

```
int SlaveManager::getSlaveFD (
            uint8_t slave_id ) const
```

Gets the file descriptor associated with the given slave ID.

Parameters

slave_id	The ID of the slave device.

Returns

The file descriptor associated with the slave device.

Definition at line 84 of file slavemanager.cpp.

7.17.3.2 getSlaveState()

```
struct sensor_packet SlaveManager::getSlaveState (
            uint8_t slave_id )
```

Retrieves the internal sensor_packet structure that contains the current state of the device.

Parameters

slave_id	The ID of the slave device.

Returns

The internal state of the device as a sensor_packet struct

Definition at line 97 of file slavemanager.cpp.

7.17.3.3 operator=() [1/2]

```
SlaveManager & SlaveManager::operator= (
            const SlaveManager &  )  [delete]
```

7.17.3.4 operator=() [2/2]

```
SlaveManager & SlaveManager::operator= (
            SlaveManager &&  )  [delete]
```

7.17.3.5 registerSlave()

```
void SlaveManager::registerSlave (
            uint8_t slave_id,
            int fd )
```

Registers a slave device with the given ID and file descriptor.

Parameters

slave_id	The ID of the slave device to register.
fd	The file descriptor associated with the slave device.

Exceptions

std::invalid_argument	if the slave ID is invalid.

Definition at line 40 of file slavemanager.cpp.

7.17.3.6 sendToSlave()

```
int SlaveManager::sendToSlave (
            uint8_t slave_id,
            const void * data,
            size_t length )
```

Sends data to the slave device with the given ID.

Parameters

slave_id	The ID of the slave device to send data to.
data	The data to send to the slave device.
length	The length of the data to send.

Returns

0 on success, -1 on failure.

Definition at line 63 of file slavemanager.cpp.

7.17.3.7 unregisterSlave()

```
void SlaveManager::unregisterSlave (
            uint8_t slave_id )
```

Unregisters a slave device with the given ID.

Parameters

slave_id	The ID of the slave device to unregister.

Exceptions

std::invalid_argument	if the slave ID is invalid.

Warning

This method closes the file descriptor associated with the slave device.

Definition at line 52 of file slavemanager.cpp.

7.17.3.8 updateSlaveState()

```
void SlaveManager::updateSlaveState (
            uint8_t slave_id,
            const struct sensor_packet & packet )
```

Updates the internal sensor_packet structure that contains the current state of the device.

Parameters

slave_id	The ID of the slave device.
packet	A sensor_packet structure that represents the updated internal state of the device.

Definition at line 93 of file slavemanager.cpp.

7.17.4 Member Data Documentation

7.17.4.1 slave_devices

```
SlaveDevice SlaveManager::slave_devices[MAX_SLAVE_ID+1]  [private]
```

Definition at line 38 of file slavemanager.h.

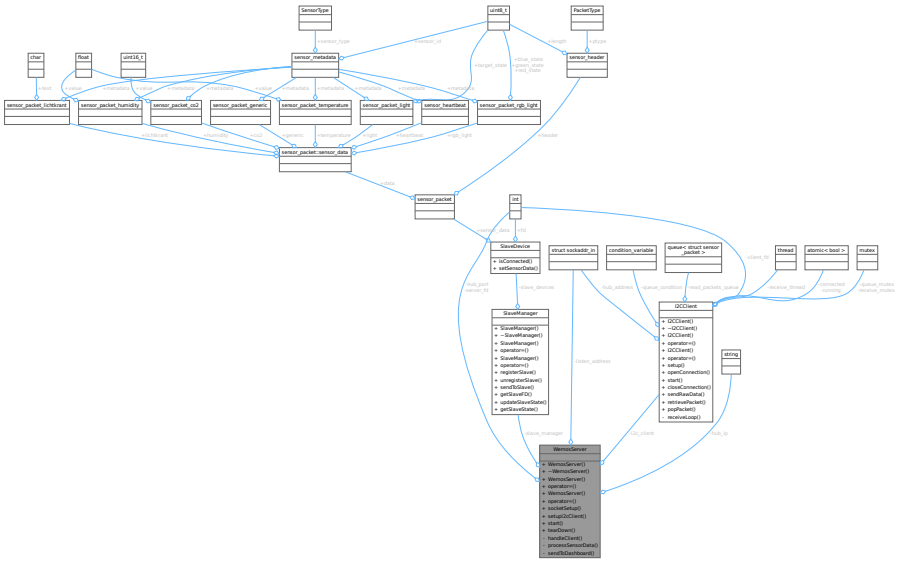
The documentation for this class was generated from the following files:

- include/slavemanager.h
- src/slavemanager.cpp

7.18 WemosServer Class Reference

```
#include <wemosserver.h>
```

Collaboration diagram for WemosServer:



Public Member Functions

- WemosServer (int port, const std::string &hub_ip, int hub_port)

  Constructor for WemosServer class.
- ~WemosServer ()
- WemosServer (const WemosServer &)=delete
- WemosServer & operator= (const WemosServer &)=delete
- WemosServer (WemosServer &&)=delete
- WemosServer & operator= (WemosServer &&)=delete
- void socketSetup ()

  Sets up the server socket and starts listening for incoming connections.
- void setupI2cClient ()

  Sets up the I2C client for communication with the I2C hub.
- void start ()
- void tearDown ()

Private Member Functions

- void handleClient (int client_fd, const struct sockaddr_in client_address)
- void processSensorData (const struct sensor_packet *data)
- void sendToDashboard (int dashboard_fd, struct sensor_packet *pkt_ptr, size_t len)

Private Attributes

- int server_fd
- struct sockaddr_in listen_address
- I2CClient i2c_client
- std::string hub_ip
- int hub_port
- SlaveManager slave_manager

7.18.1 Detailed Description

Definition at line 20 of file wemosserver.h.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 WemosServer() [1/3]

```
WemosServer::WemosServer (
            int port,
            const std::string & hub_ip,
            int hub_port )
```

Constructor for WemosServer class.

This constructor initializes the server with the specified port, hub IP address, and hub port.

Parameters

port	The port number on which the server will listen for incoming connections.
hub_ip	The IP address of the I2C hub.
hub_port	The port number of the I2C hub.

Exceptions

std::invalid_argument	if the port number is invalid.

Warning

This constructor does not start the server loop. The loop() method should be called separately to start accepting client connections.

Definition at line 202 of file wemosserver.cpp.

7.18.2.2 ~WemosServer()

```
WemosServer::~WemosServer ( )
```

Definition at line 216 of file wemosserver.cpp.

7.18.2.3 WemosServer() [2/3]

```
WemosServer::WemosServer (
            const WemosServer &  )  [delete]
```

7.18.2.4 WemosServer() [3/3]

```
WemosServer::WemosServer (
            WemosServer &&  )  [delete]
```

7.18.3 Member Function Documentation

7.18.3.1 handleClient()

```
void WemosServer::handleClient (
            int client_fd,
            const struct sockaddr_in client_address )  [private]
```

Definition at line 39 of file wemosserver.cpp.

7.18.3.2 operator=() [1/2]

```
WemosServer & WemosServer::operator= (
            const WemosServer &  )  [delete]
```

7.18.3.3 operator=() [2/2]

```
WemosServer & WemosServer::operator= (
            WemosServer &&  )  [delete]
```

7.18.3.4 processSensorData()

```
void WemosServer::processSensorData (
            const struct sensor_packet * data )  [private]
```

Definition at line 154 of file wemosserver.cpp.

7.18.3.5 sendToDashboard()

```
void WemosServer::sendToDashboard (
            int dashboard_fd,
            struct sensor_packet * pkt_ptr,
            size_t len )  [private]
```

Definition at line 197 of file wemosserver.cpp.

7.18.3.6 setupI2cClient()

```
void WemosServer::setupI2cClient ( )
```

Sets up the I2C client for communication with the I2C hub.

Definition at line 252 of file wemosserver.cpp.

7.18.3.7 socketSetup()

```
void WemosServer::socketSetup ( )
```

Sets up the server socket and starts listening for incoming connections.

This method creates a socket, binds it to the specified port, and starts listening for incoming client connections. It also sets the socket options to allow address reuse.

Exceptions

std::runtime_error	if socket creation, binding, or listening fails.

Warning

This metho d should be called before starting the server loop.

Definition at line 221 of file wemosserver.cpp.

7.18.3.8 start()

```
void WemosServer::start ( )
```

Definition at line 254 of file wemosserver.cpp.

7.18.3.9 tearDown()

```
void WemosServer::tearDown ( )
```

Definition at line 292 of file wemosserver.cpp.

7.18.4 Member Data Documentation

7.18.4.1 hub_ip

```
std::string WemosServer::hub_ip  [private]
```

Definition at line 26 of file wemosserver.h.

7.18.4.2 hub_port

```
int WemosServer::hub_port  [private]
```

Definition at line 27 of file wemosserver.h.

7.18.4.3 i2c_client

```
I2CClient WemosServer::i2c_client  [private]
```

Definition at line 25 of file wemosserver.h.

7.18.4.4 listen_address

```
struct sockaddr_in WemosServer::listen_address  [private]
```

Definition at line 23 of file wemosserver.h.

7.18.4.5 server_fd

```
int WemosServer::server_fd  [private]
```

Definition at line 22 of file wemosserver.h.

7.18.4.6 slave_manager

```
SlaveManager WemosServer::slave_manager  [private]
```

Definition at line 29 of file wemosserver.h.

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

- include/wemosserver.h
- src/wemosserver.cpp

Chapter 8

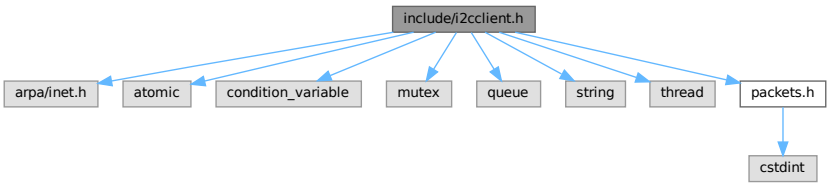
File Documentation

8.1 include/i2cclient.h File Reference

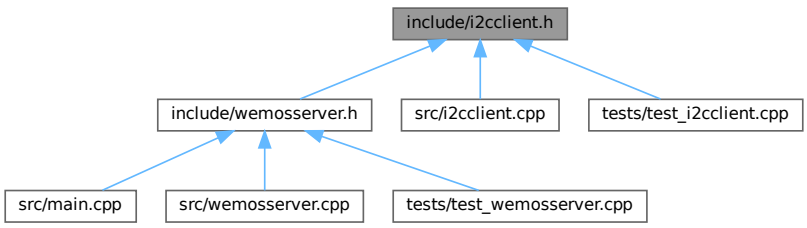
Header file for i2cclient.cpp.

```
#include <arpa/inet.h>
#include <atomic>
#include <condition_variable>
#include <mutex>
#include <queue>
#include <string>
#include <thread>
#include "packets.h"
```

Include dependency graph for i2cclient.h:



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



Classes

- class I2CClient
- struct I2CClient::DataReceiveReturn

8.1.1 Detailed Description

Header file for i2cclient.cpp.

This file contains declarations for the classes and functions used in the Wemos server application.

Author

Daan Breur

Erynn Scholtes

Definition in file i2cclient.h.

8.2 i2cclient.h

Go to the documentation of this file.

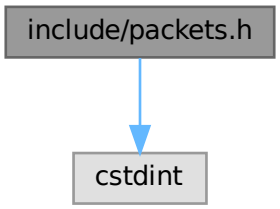
```
00001
00010 #ifndef I2CCLIENT_H
00011 #define I2CCLIENT_H
00012
00013 #include <arpa/inet.h>
00014
00015 #include <atomic>
00016 #include <condition_variable>
00017 #include <mutex>
00018 #include <queue>
00019 #include <string>
00020 #include <thread>
00021
00022 #include "packets.h"
00023
00024 class I2CClient {
00025   private:
00026     int client_fd;
00027
00028     struct sockaddr_in hub_address;
00029
00030     std::thread receive_thread;
00031
00032     std::atomic<bool> connected;
00033     std::atomic<bool> running;
00034
00035     std::mutex receive_mutex;
00036     std::mutex queue_mutex;
00037
00038     std::condition_variable queue_condition;
00039
00040     std::queue<struct sensor_packet> read_packets_queue;
00041
00049     void receiveLoop();
00050
00051   public:
00052     struct DataReceiveReturn {
00053         uint8_t *data;
00054         size_t length;
00055     };
00056
00057   public:
00065     I2CClient();
00066     ~I2CClient();
00067
00068     I2CClient(const I2CClient &) = delete;
00069     I2CClient &operator=(const I2CClient &) = delete;
00070     I2CClient(I2CClient &&) = delete;
```

```
00071     I2CClient &operator=(I2CClient &&) = delete;
00072
00081     void setup(const std::string &ip, int port);
00082
00089     bool openConnection();
00090
00097     void start();
00098
00103     void closeConnection();
00104
00111     void sendRawData(uint8_t *data, size_t length);
00112
00118     // void sendData();
00119
00126     struct sensor_packet retrievePacket(bool block = false);
00127
00128     struct sensor_packet popPacket();
00129 };
00130
00131 #endif
```

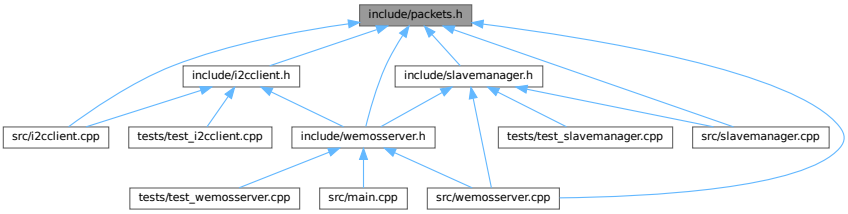
8.3 include/packets.h File Reference

Header file for packets.h.

`#include <cstdint>`
Include dependency graph for packets.h:



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



Classes

- struct sensor_header

Header structure for sensor packets.

- struct sensor_metadata

  Structure for sensor metadata, which is always included in any packet.
- struct sensor_heartbeat

  Structure for heartbeat packets.
- struct sensor_packet_generic

  Structure for generic sensor packets.
- struct sensor_packet_temperature

  Structure for temperature sensor packets.
- struct sensor_packet_co2

  Structure for CO2 sensor packets.
- struct sensor_packet_humidity

  Structure for humidity sensor packets.
- struct sensor_packet_light

  Structure for light sensor packets.
- struct sensor_packet_rgb_light

  Structure for RGB light sensor packets.
- struct sensor_packet_lichtkrant

  structure for the lichkrant packets
- struct sensor_packet

  Union structure for the entire sensor packet.
- union sensor_packet::sensor_data

- union sensor_data

Enumerations

- enum class SensorType : uint8_t {
  NOOP = 0 , BUTTON = 1 , TEMPERATURE = 2 , CO2 = 3 ,
  HUMIDITY = 4 , PRESSURE = 5 , LIGHT = 6 , MOTION = 7 ,
  RGB_LIGHT = 8 , LICHTKRANT = 9 }
- enum class PacketType : uint8_t {
  DATA = 0 , HEARTBEAT = 1 , DASHBOARD_POST = 2 , DASHBOARD_GET = 3 ,
  DASHBOARD_RESPONSE = 4 }

Functions

- struct sensor_header __attribute__ ((packed))

Variables

- uint8_t length

  Length of the packet excluding the header.
- PacketType ptype

  Type of the packet as PacketType (DATA, HEARTBEAT, etc.).
- SensorType sensor_type

  Type of the sensor being addressed as SensorType (one byte)

- uint8_t sensor_id

  ID of the sensor being addressed.
- struct sensor_metadata metadata
- float value

  Value of the sensor reading the temperature represented in Celcius.
- uint8_t target_state

  Target state of the light (on 1/off 0) represented as a boolean value.
- uint8_t red_state

  Target state of the red color (0-255) represented as an 8-bit integer.
- uint8_t green_state

  Target state of the green color (0-255) represented as an 8-bit integer.
- uint8_t blue_state

  Target state of the blue color (0-255) represented as an 8-bit integer.
- char text [16]
- struct sensor_header header

  Header of the packet containing length and type information.
- union sensor_data data

8.3.1 Detailed Description

Header file for packets.h.

This files origin is from the Wemos project

Warning

THIS FILE MUST BE KEPT IN SYNC IN OTHER PROJECTS

Author

Daan Breur

Erynn Scholtes

Definition in file packets.h.

8.3.2 Enumeration Type Documentation

8.3.2.1 PacketType

```
enum class PacketType :  uint8_t  [strong]
```

Enumerator

DATA	
HEARTBEAT	
DASHBOARD_POST	
DASHBOARD_GET	
DASHBOARD_RESPONSE	

Definition at line 28 of file packets.h.

8.3.2.2 SensorType

```
enum class SensorType :  uint8_t  [strong]
```

Enumerator

NOOP	
BUTTON	
TEMPERATURE	
CO2	
HUMIDITY	
PRESSURE	
LIGHT	
MOTION	
RGB_LIGHT	
LICHTKRANT	

Definition at line 15 of file packets.h.

8.3.3 Function Documentation

8.3.3.1 __attribute__()

```
struct sensor_packet __attribute__ (
             (packed)  )
```

8.3.4 Variable Documentation

8.3.4.1 blue_state

```
uint8_t blue_state
```

Target state of the blue color (0-255) represented as an 8-bit integer.

Definition at line 6 of file packets.h.

8.3.4.2 data

```
union sensor_data data
```

8.3.4.3 green_state

```
uint8_t green_state
```

Target state of the green color (0-255) represented as an 8-bit integer.

Definition at line 4 of file packets.h.

8.3.4.4 header

```
struct sensor_header header
```

Header of the packet containing length and type information.

Definition at line 1 of file packets.h.

8.3.4.5 length

```
uint8_t length
```

Length of the packet excluding the header.

Definition at line 1 of file packets.h.

8.3.4.6 metadata

```
struct sensor_metadata metadata
```

Definition at line 0 of file packets.h.

8.3.4.7 ptype

```
PacketType ptype
```

Type of the packet as PacketType (DATA, HEARTBEAT, etc.).

Definition at line 3 of file packets.h.

8.3.4.8 red_state

```
uint8_t red_state
```

Target state of the red color (0-255) represented as an 8-bit integer.

Definition at line 2 of file packets.h.

8.3.4.9 sensor_id

```
uint8_t sensor_id
```

ID of the sensor being addressed.

Definition at line 3 of file packets.h.

8.3.4.10 sensor_type

```
SensorType sensor_type
```

Type of the sensor being addressed as SensorType (one byte)

Definition at line 1 of file packets.h.

8.3.4.11 target_state

```
uint8_t target_state
```

Target state of the light (on 1/off 0) represented as a boolean value.

Definition at line 2 of file packets.h.

8.3.4.12 text

```
char text[16]
```

Definition at line 1 of file packets.h.

8.3.4.13 value

```
float value
```

Value of the sensor reading the temperature represented in Celcius.

Value of the sensor reading the humidity level represented in percentage.

Value of the sensor reading the CO2 level represented in ppm.

Definition at line 2 of file packets.h.

8.4 packets.h

Go to the documentation of this file.

```
00001
00010 #ifndef PACKETS_H
00011 #define PACKETS_H
00012
00013 #include <cstdint>
00014
00015 enum class SensorType : uint8_t {
00016     NOOP = 0,
00017     BUTTON = 1,
00018     TEMPERATURE = 2,
00019     CO2 = 3,
00020     HUMIDITY = 4,
00021     PRESSURE = 5,
00022     LIGHT = 6,
00023     MOTION = 7,
00024     RGB_LIGHT = 8,
00025     LICHTKRANT = 9,
00026 };
00027
00028 enum class PacketType : uint8_t {
00029     DATA = 0,
00030     HEARTBEAT = 1,
00031     DASHBOARD_POST = 2,
00032     DASHBOARD_GET = 3,
00033     DASHBOARD_RESPONSE = 4
00034 };
00035
00041 struct sensor_header {
00043     uint8_t length;
00045     PacketType ptype;
00046 } __attribute__((packed));
00047
00053 struct sensor_metadata {
00055     SensorType sensor_type;
00057     uint8_t sensor_id;
00058 } __attribute__((packed));
00059
00060 // Specific packet structures (ensure alignment/packing matches expected format)
00061
00070 struct sensor_heartbeat {
00071     struct sensor_metadata metadata;
00072 } __attribute__((packed));
00073
00082 struct sensor_packet_generic {
00083     struct sensor_metadata metadata;
00084     // /** @brief Whether the sensor did or did not trigger */
00085     // bool value;
00086 } __attribute__((packed));
00087
00095 struct sensor_packet_temperature {
00096     struct sensor_metadata metadata;
00098     float value;
00099 } __attribute__((packed));
00100
00108 struct sensor_packet_co2 {
00109     struct sensor_metadata metadata;
00111     uint16_t value;
00112 } __attribute__((packed));
00113
00121 struct sensor_packet_humidity {
00122     struct sensor_metadata metadata;
00124     float value;
00125 } __attribute__((packed));
00126
00134 struct sensor_packet_light {
00135     struct sensor_metadata metadata;
00137     uint8_t target_state;
00138 } __attribute__((packed));
00139
00148 struct sensor_packet_rgb_light {
00149     struct sensor_metadata metadata;
00151     uint8_t red_state;
00153     uint8_t green_state;
00155     uint8_t blue_state;
00156 } __attribute__((packed));
00157
00164 struct sensor_packet_lichtkrant {
00165     struct sensor_metadata metadata;
00166     char text[16];
00167 } __attribute__((packed));
00168 // --- End Structures ---
00169
```

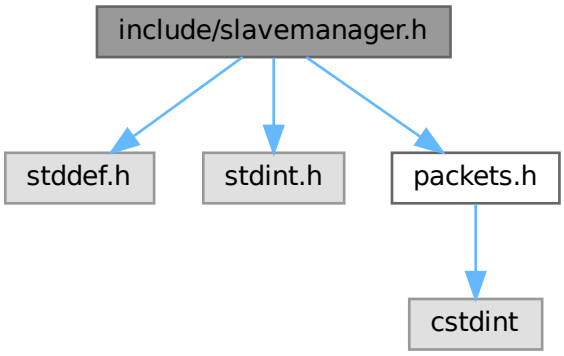
```
00234 struct sensor_packet {
00236     struct sensor_header header;
00237
00239     union sensor_data {
00240         struct sensor_heartbeat heartbeat;
00241         struct sensor_packet_generic generic;
00242         struct sensor_packet_temperature temperature;
00243         struct sensor_packet_co2 co2;
00244         struct sensor_packet_humidity humidity;
00245         struct sensor_packet_light light;
00246         struct sensor_packet_rgb_light rgb_light;
00247         struct sensor_packet_lichtkrant lichtkrant;
00248     } data;
00249 } __attribute__((packed));
00250
00251 #endif  // PACKETS_H
```

8.5 include/slavemanager.h File Reference

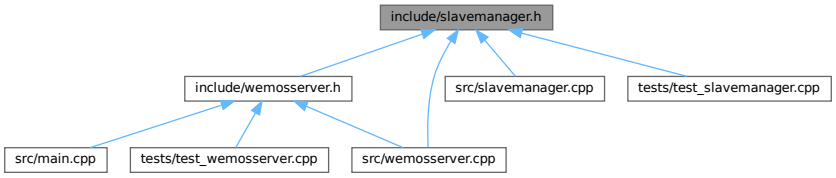
Header file for slavemanager.cpp.

```
#include <stddef.h>
#include <stdint.h>
#include "packets.h"
```

Include dependency graph for slavemanager.h:



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



Classes

- struct SlaveDevice

  Structure representing a slave device.
- class SlaveManager

Macros

- #define MAX_SLAVE_ID 0xFF

  Biggest possible slave ID.

8.5.1 Detailed Description

Header file for slavemanager.cpp.

This file contains declarations for the SlaveManager class and the SlaveDevice struct. The SlaveManager class is responsible for managing slave devices and their file descriptors.

Author

Daan Breur

Definition in file slavemanager.h.

8.5.2 Macro Definition Documentation

8.5.2.1 MAX_SLAVE_ID

```
#define MAX_SLAVE_ID 0xFF
```

Biggest possible slave ID.

Definition at line 16 of file slavemanager.h.

8.6 slavemanager.h

Go to the documentation of this file.

```
00001
00010 #ifndef SLAVEMANAGER_H
00011 #define SLAVEMANAGER_H
00012
00016 #define MAX_SLAVE_ID 0xFF
00017
00018 #include <stddef.h>
00019 #include <stdint.h>
00020
00021 #include "packets.h"
00022
00028 struct SlaveDevice {
00029     int fd = -1;
00030     struct sensor_packet sensor_data = {0};
00031
00032     bool isConnected() const;
00033     void setSensorData(const struct sensor_packet &);
00034 };
```

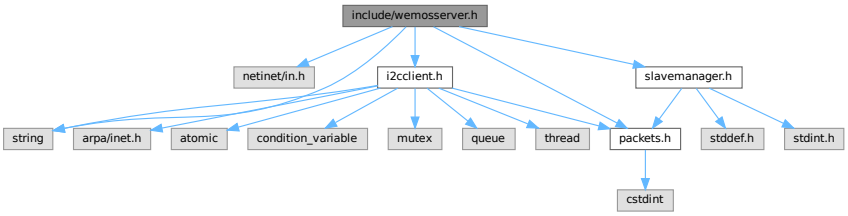
```
00035
00036 class SlaveManager {
00037    private:
00038     SlaveDevice slave_devices[MAX_SLAVE_ID + 1];
00039
00040    public:
00041     SlaveManager();
00042     ~SlaveManager();
00043
00044     SlaveManager(const SlaveManager &) = delete;
00045     SlaveManager &operator=(const SlaveManager &) = delete;
00046     SlaveManager(SlaveManager &&) = delete;
00047     SlaveManager &operator=(SlaveManager &&) = delete;
00048
00055     void registerSlave(uint8_t slave_id, int fd);
00056
00063     void unregisterSlave(uint8_t slave_id);
00064
00072     int sendToSlave(uint8_t slave_id, const void *data, size_t length);
00073
00079     int getSlaveFD(uint8_t slave_id) const;
00080
00088     void updateSlaveState(uint8_t slave_id, const struct sensor_packet &packet);
00089
00096     struct sensor_packet getSlaveState(uint8_t slave_id);
00097 };
00098
00099 #endif
```

8.7 include/wemosserver.h File Reference

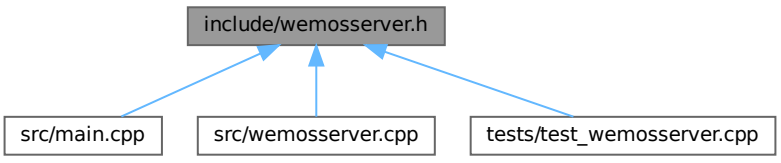
Header file for wemosserver.cpp.

```
#include <netinet/in.h>
#include <string>
#include "i2cclient.h"
#include "packets.h"
#include "slavemanager.h"
```

Include dependency graph for wemosserver.h:



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



Classes

- class WemosServer

8.7.1 Detailed Description

Header file for wemosserver.cpp.

This file contains declarations for the classes and functions used in the Wemos server application.

Author

Daan Breur

Definition in file wemosserver.h.

8.8 wemosserver.h

Go to the documentation of this file.

```
00001
00009 #ifndef WEMOSSERVER_H
00010 #define WEMOSSERVER_H
00011
00012 #include <netinet/in.h>
00013
00014 #include <string>
00015
00016 #include "i2cclient.h"
00017 #include "packets.h"
00018 #include "slavemanager.h"
00019
00020 class WemosServer {
00021    private:
00022     int server_fd;
00023     struct sockaddr_in listen_address;
00024
00025     I2CClient i2c_client;
00026     std::string hub_ip;
00027     int hub_port;
00028
00029     SlaveManager slave_manager;
00030
00031     void handleClient(int client_fd, const struct sockaddr_in client_address);
00032
00033     void processSensorData(const struct sensor_packet *data);
00034
00035     void sendToDashboard(int dashboard_fd, struct sensor_packet *pkt_ptr, size_t len);
00036
00037    public:
00049     WemosServer(int port, const std::string &hub_ip, int hub_port);
00050     ~WemosServer();
00051
00052     WemosServer(const WemosServer &) = delete;
00053     WemosServer &operator=(const WemosServer &) = delete;
00054     WemosServer(WemosServer &&) = delete;
00055     WemosServer &operator=(WemosServer &&) = delete;
00056
00065     void socketSetup();
00066
00070     void setupI2cClient();
00071
00072     void start();
00073
00074     void tearDown();
00075 };
00076
00077 #endif
```

8.9 modules.dox File Reference

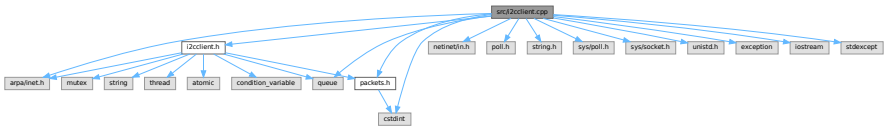
8.10 README.md File Reference

8.11 src/i2cclient.cpp File Reference

Implementation of I2CClient class.

```
#include "i2cclient.h"
#include <arpa/inet.h>
#include <netinet/in.h>
#include <poll.h>
#include <string.h>
#include <sys/poll.h>
#include <sys/socket.h>
#include <unistd.h>
#include <cstdint>
#include <exception>
#include <iostream>
#include <queue>
#include <stdexcept>
#include "packets.h"
```

Include dependency graph for i2cclient.cpp:



Macros

- #define BUFFER_SIZE 1024
- #define THREAD_RELINQUISH(mut)

8.11.1 Detailed Description

Implementation of I2CClient class.

Author

Daan Breur

Erynn Scholtes

Definition in file i2cclient.cpp.

8.11.2 Macro Definition Documentation

8.11.2.1 BUFFER_SIZE

```
#define BUFFER_SIZE 1024
```

Definition at line 26 of file i2cclient.cpp.

8.11.2.2 THREAD_RELINQUISH

```
#define THREAD_RELINQUISH(
              mut )
```

Value:

```
    {                           \
        mut.unlock();           \
        continue;               \
    }
```

Definition at line 39 of file i2cclient.cpp.

8.12 i2cclient.cpp

Go to the documentation of this file.

```
00001
00008 #include "i2cclient.h"
00009
00010 #include <arpa/inet.h>
00011 #include <netinet/in.h>
00012 #include <poll.h>
00013 #include <string.h>
00014 #include <sys/poll.h>
00015 #include <sys/socket.h>
00016 #include <unistd.h>
00017
00018 #include <cstdint>
00019 #include <exception>
00020 #include <iostream>
00021 #include <queue>
00022 #include <stdexcept>
00023
00024 #include "packets.h"
00025
00026 #define BUFFER_SIZE 1024
00027
00028 I2CClient::I2CClient() : client_fd(-1), connected(false), running(false) {
00029     memset(&hub_address, 0, sizeof(hub_address));
00030 }
00031
00032 I2CClient::~I2CClient() {
00033     if (connected) closeConnection();
00034 }
00035
00036 // first unlocks the mutex passed, then continues in the while loop
00037 // WARNING: ! only use when the mutex is currently locked !
00038 // just "continue;" otherwise
00039 #define THREAD_RELINQUISH(mut) \
00040     {                          \
00041         mut.unlock();          \
00042         continue;              \
00043     }
00044
00045 void I2CClient::receiveLoop() {
00046     uint8_t receive_buffer[BUFFER_SIZE] = {0};
00047     struct pollfd pf;
00048
00049     pf.fd = client_fd;
00050     pf.events = POLLIN;
00051
00052     while (true == running && true == connected) {
```

```
00053         // TODO: revise error handling within the loop;
00054         // maybe always stop loop on error, instead of just continuing?
00055         // problem for another time :clueless:
00056         // - Erynn
00057
00058         receive_mutex.lock();
00059
00060         // poll offers an easy way to wait up to one second between iterations
00061         int sockets_ready = poll(&pf, 1, 1000);
00062
00063         if (sockets_ready < 1) {
00064             // something went wrong
00065             if (sockets_ready == -1) perror("poll() failed");  // error happened, else timeout
00066
00067             THREAD_RELINQUISH(receive_mutex);
00068         }
00069
00070         // if we get here, there is guaranteed to be readable data.
00071         // either this data is because the other end disconnected, or because there
00072         // is proper data to read from the wire
00073         int amount_read = recv(pf.fd, receive_buffer, BUFFER_SIZE, MSG_DONTWAIT);
00074
00075         if (amount_read == -1) {
00076             // error occured, errno set
00077             perror("recv() failed");
00078
00079             THREAD_RELINQUISH(receive_mutex);
00080         } else if (amount_read == 0) {
00081             // socket disconnected
00082             connected = false;
00083             running = false;
00084             client_fd = -1;
00085
00086             THREAD_RELINQUISH(receive_mutex);
00087         }
00088
00089         receive_mutex.unlock();
00090
00091         std::cout « "Received " « amount_read « " bytes from Raspberry PI I2C controller."
00092                   « std::endl;
00093
00094         for (int i = 0; i < amount_read; ++i) {
00095             printf("%02X ", receive_buffer[i]);
00096         }
00097         printf("\n");
00098
00099         {
00100             size_t buffer_offset = 0;
00101
00102             do {
00103                 const struct sensor_header *head =
00104                     (const struct sensor_header *)&receive_buffer[buffer_offset];
00105                 uint8_t length = head->length;
00106                 uint8_t p_type = (uint8_t)head->ptype;
00107
00108                 uint8_t s_type = receive_buffer[sizeof(*head)];
00109
00110                 if ((buffer_offset + sizeof(*head) + length) > amount_read) {
00111                     // oopsie woopsie; incomplete packet from RPI
00112                     printf(
00113                         "We received an incomplete packet from the Raspberry Pi I2C controller; "
00114                         "Discarding...\n");
00115                     break;
00116                 }
00117
00118                 struct sensor_packet packet = {0};
00119                 int to_copy = sizeof(*head) + length;
00120                 if (to_copy > sizeof(packet)) to_copy = sizeof(packet);
00121                 memcpy(&packet, receive_buffer + buffer_offset, to_copy);
00122
00123                 printf("AAAAAAAAAAAAA\n");
00124                 queue_mutex.lock();
00125                 read_packets_queue.push(packet);
00126                 queue_mutex.unlock();
00127                 queue_condition.notify_one();  // maybe switch this with the line before if issues
00128                                                // occur - Erynn
00129
00130                 buffer_offset += sizeof(*head) + length;
00131             } while (buffer_offset + sizeof(struct sensor_header) <= amount_read);
00132         }
00133     }
00134
00135     std::terminate();
00136 }
00137
00138 void I2CClient::setup(const std::string &hub_ip, int hub_port) {
00139     if (inet_pton(AF_INET, hub_ip.c_str(), &hub_address.sin_addr) <= 0) {
```

```
00140         perror("inet_pton()");
00141         throw std::invalid_argument("Invalid IP address");
00142     }
00143
00144     if (hub_port <= 0 || hub_port > 65535) throw std::invalid_argument("Invalid port number");
00145
00146     hub_address.sin_family = AF_INET;
00147     hub_address.sin_port = htons(hub_port);
00148 }
00149
00150 bool I2CClient::openConnection() {
00151     if (client_fd >= 0) {
00152         close(client_fd);
00153         client_fd = -1;
00154     }
00155     connected = false;
00156
00157     std::string ip(inet_ntoa(hub_address.sin_addr));
00158     uint16_t port = ntohs(hub_address.sin_port);
00159
00160     std::cout « "Connecting to I2C hub at " « ip « ":" « port « std::endl;
00161
00162     client_fd = socket(AF_INET, SOCK_STREAM, 0);
00163     if (client_fd < 0) {
00164         std::cerr « "Socket creation failed" « std::endl;
00165         return false;
00166     }
00167
00168     if (connect(client_fd, (struct sockaddr *)&hub_address, sizeof(hub_address)) < 0) {
00169         int err = errno;
00170         std::cerr « "Connection failed: " « strerror(err) « std::endl;
00171         close(client_fd);
00172         client_fd = -1;
00173         return false;
00174     }
00175
00176     std::cout « "Connected to I2C hub at " « ip « ":" « port « std::endl;
00177     connected = true;
00178
00179     return true;
00180 }
00181
00182 void I2CClient::start() {
00183     if (!connected) {
00184         std::cerr
00185             « "Could not start communicating with I2C-bridge because not connected to I2C hub"
00186             « std::endl;
00187         throw std::runtime_error("Not connected to I2C-bridge");
00188     }
00189
00190     running = true;
00191     receive_thread = std::thread(&I2CClient::receiveLoop, this);
00192 }
00193
00194 void I2CClient::closeConnection() {
00195     if (!connected) {
00196         std::cerr « "Could not close the connection to I2C-bridge because not connected to I2C "
00197                     "hub (either already closed, or never connected in the first place)"
00198                   « std::endl;
00199         return;
00200     }
00201
00202     running = false;
00203     receive_thread.join();
00204 }
00205
00206 void I2CClient::sendRawData(uint8_t *data, size_t length) {
00207     if (send(client_fd, data, length, 0) == -1) {
00208         perror("send() failed");
00209         throw std::runtime_error("Sending data to I2C-bridge failed");
00210     }
00211 }
00212
00213 struct sensor_packet I2CClient::retrievePacket(bool block) {
00214     // if (read_packets_queue.size() < 1 && !block) {
00215     //     // there are no packets available to retrieve,
00216     //     // and we're not blocking
00217     //     throw std::runtime_error("No packet data available to retrieve from I2C-bridge");
00218     // } else if (read_packets_queue.size() < 1) {
00219     //     // there are no packets available, but we block until there is
00220     //     std::unique_lock lk(queue_mutex);
00221     //     printf("waiting\n");
00222     //     queue_condition.wait(lk);
00223     //     printf("done waiting\n");
00224     // }
00225
00226     struct sensor_packet pkt = {0};
```

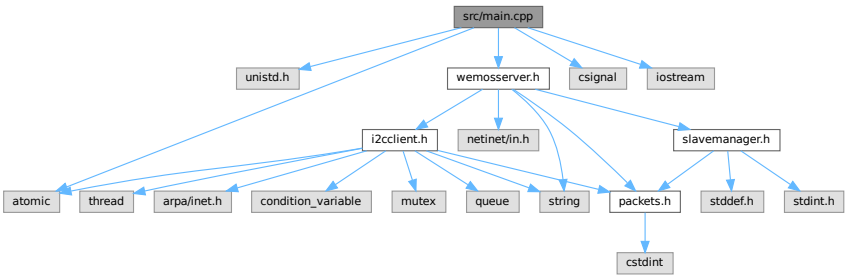
```
00227     while (read_packets_queue.size() < 1) {
00228         if (!block) return pkt;
00229         usleep(1000);
00230     }
00231
00232     struct sensor_packet return_packet;
00233     memcpy(&return_packet, &read_packets_queue.front(), sizeof(return_packet));
00234     read_packets_queue.pop();
00235
00236     printf("packet get\n");
00237
00238     printf("returning\n");
00239     return return_packet;
00240 }
```

8.13 src/main.cpp File Reference

Main entrypoint for Wemos Bridge Server application.

```
#include <unistd.h>
#include <atomic>
#include <csignal>
#include <iostream>
#include "wemosserver.h"
```

Include dependency graph for main.cpp:



Macros

- #define SERVER_PORT 5000
- #define I2C_HUB_IP "192.168.226.245"
- #define I2C_HUB_PORT 5000

Functions

- std::atomic< bool > global_shutdown_flag (false)
- void signalHandler (int signum)
- int main ()

8.13.1 Detailed Description

Main entrypoint for Wemos Bridge Server application.

Definition in file main.cpp.

8.13.2 Macro Definition Documentation

8.13.2.1 I2C_HUB_IP

```
#define I2C_HUB_IP "192.168.226.245"
```

Definition at line 15 of file main.cpp.

8.13.2.2 I2C_HUB_PORT

```
#define I2C_HUB_PORT 5000
```

Definition at line 16 of file main.cpp.

8.13.2.3 SERVER_PORT

```
#define SERVER_PORT 5000
```

Definition at line 14 of file main.cpp.

8.13.3 Function Documentation

8.13.3.1 global_shutdown_flag()

```
std::atomic< bool > global_shutdown_flag (
            false  )
```

8.13.3.2 main()

```
int main ( )
```

Definition at line 27 of file main.cpp.

8.13.3.3 signalHandler()

```
void signalHandler (
            int signum )
```

Definition at line 20 of file main.cpp.

8.14 main.cpp

Go to the documentation of this file.

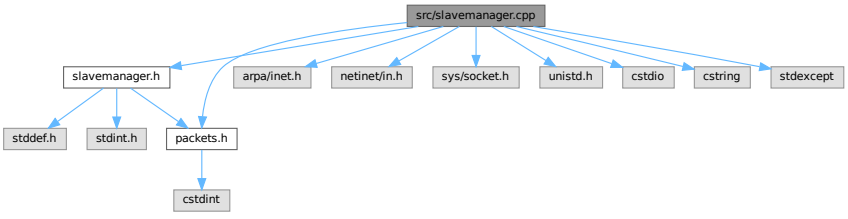
```
00001
00006 #include <unistd.h>
00007
00008 #include <atomic>
00009 #include <csignal>
00010 #include <iostream>
00011
00012 #include "wemosserver.h"
00013
00014 #define SERVER_PORT 5000
00015 #define I2C_HUB_IP "192.168.226.245"
00016 #define I2C_HUB_PORT 5000
00017
00018 std::atomic<bool> global_shutdown_flag(false);
00019
00020 void signalHandler(int signum) {
00021     std::cout « "Interrupt signal (" « signum « ") received.\n";
00022     if (signum == SIGINT || signum == SIGTERM) {
00023         global_shutdown_flag = true;
00024     }
00025 }
00026
00027 int main() {
00028     setbuf(stdout, NULL);
00029     std::cout « "Starting Wemos Bridge on port " « SERVER_PORT « std::endl;
00030
00031     // signal(SIGINT, signalHandler);
00032     // signal(SIGTERM, signalHandler);
00033
00034     WemosServer server(SERVER_PORT, I2C_HUB_IP, I2C_HUB_PORT);
00035
00036     sleep(1);
00037
00038     server.start();
00039
00040     return 0;
00041 }
```

8.15 src/slavemanager.cpp File Reference

Implementation of SlaveManager class.

```
#include "slavemanager.h"
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <unistd.h>
#include <cstdio>
#include <cstring>
#include <stdexcept>
#include "packets.h"
```

Include dependency graph for slavemanager.cpp:



8.15.1 Detailed Description

Implementation of SlaveManager class.

Author

Daan Breur

Definition in file slavemanager.cpp.

8.16 slavemanager.cpp

Go to the documentation of this file.

```
00001
00007 #include "slavemanager.h"
00008
00009 #include <arpa/inet.h>
00010 #include <netinet/in.h>
00011 #include <sys/socket.h>
00012 #include <unistd.h>
00013
00014 #include <cstdio>
00015 #include <cstring>
00016 #include <stdexcept>
00017
00018 #include "packets.h"
00019
00020 bool SlaveDevice::isConnected() const { return (-1 != fd); }
00021 void SlaveDevice::setSensorData(const struct sensor_packet& pkt) {
00022     memcpy(&sensor_data, &pkt, sizeof(sensor_data));
00023 }
00024
00025 SlaveManager::SlaveManager() {
00026     for (int i = 0; i <= MAX_SLAVE_ID; ++i) {
00027         slave_devices[i].fd = -1;
00028     }
00029 }
00030
00031 SlaveManager::~SlaveManager() {
00032     for (int i = 0; i <= MAX_SLAVE_ID; ++i) {
00033         if (slave_devices[i].fd >= 0) {
00034             close(slave_devices[i].fd);
00035             slave_devices[i].fd = -1;
00036         }
00037     }
00038 }
00039
00040 void SlaveManager::registerSlave(uint8_t slave_id, int fd) {
00041     if (slave_id > MAX_SLAVE_ID || slave_id < 0) {
00042         printf("Invalid slave ID=%u\n", slave_id);
00043         throw std::invalid_argument("Invalid slave ID");
00044     }
00045
00046     printf("Registering new slave ID=%u\n", slave_id);
00047
00048     slave_devices[slave_id].fd = fd;
00049     memset(&slave_devices[slave_id].sensor_data, 0, sizeof(slave_devices[slave_id].sensor_data));
00050 }
00051
00052 void SlaveManager::unregisterSlave(uint8_t slave_id) {
00053     if (slave_id > MAX_SLAVE_ID || slave_id < 0) {
00054         printf("Invalid slave ID=%u\n", slave_id);
00055         throw std::invalid_argument("Invalid slave ID");
00056     }
00057
00058     printf("Unregistering slave ID=%u\n", slave_id);
00059     close(slave_devices[slave_id].fd);
00060     slave_devices[slave_id].fd = -1;
00061 }
00062
00063 int SlaveManager::sendToSlave(uint8_t slave_id, const void* data, size_t length) {
00064     if (slave_id > MAX_SLAVE_ID || slave_id < 0) {
00065         printf("Invalid slave ID=%u\n", slave_id);
00066         throw std::invalid_argument("Invalid slave ID");
00067     }
```

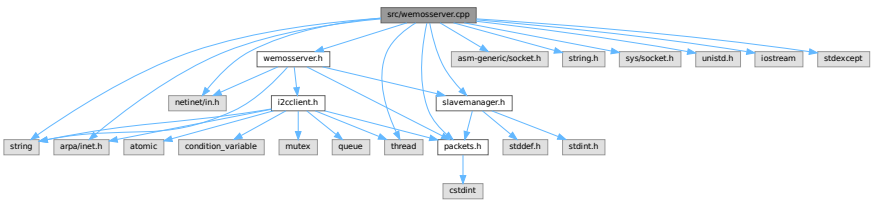
```
00068
00069     printf("Sending %zu bytes to slave ID=%u\n", length, slave_id);
00070     if (slave_devices[slave_id].fd < 0) {
00071         printf("Slave ID=%u not registered\n", slave_id);
00072         return -1;
00073     }
00074
00075     ssize_t bytes_sent = send(slave_devices[slave_id].fd, data, length, 0);
00076     if (bytes_sent < 0) {
00077         perror("send to slave failed");
00078         return -1;
00079     }
00080
00081     return 0;
00082 }
00083
00084 int SlaveManager::getSlaveFD(uint8_t slave_id) const {
00085     if (slave_id > MAX_SLAVE_ID || slave_id < 0) {
00086         printf("Invalid slave ID=%u\n", slave_id);
00087         throw std::invalid_argument("Invalid slave ID");
00088     }
00089
00090     return slave_devices[slave_id].fd;
00091 }
00092
00093 void SlaveManager::updateSlaveState(uint8_t slave_id, const struct sensor_packet& packet) {
00094     slave_devices[slave_id].setSensorData(packet);
00095 }
00096
00097 struct sensor_packet SlaveManager::getSlaveState(uint8_t slave_id) {
00098     return slave_devices[slave_id].sensor_data;
00099 }
```

8.17 src/wemosserver.cpp File Reference

Implementation of WemosServer class.

```
#include "wemosserver.h"
#include <arpa/inet.h>
#include <asm-generic/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#include <iostream>
#include <stdexcept>
#include <string>
#include <thread>
#include "packets.h"
#include "slavemanager.h"
```

Include dependency graph for wemosserver.cpp:



Macros

- #define BUFFER_SIZE 1024

Maximum data size to be read or sent over the wire.

- #define MAX_CLIENTS 128

  Maximum number of clients that can be connected to the server.
- #define TAFEL_KNOP_1 0x80
- #define TAFEL_LAMP_1 0x6D

8.17.1 Detailed Description

Implementation of WemosServer class.

This file contains the implementation of the WemosServer class, which handles the server functionality for the Wemos device. It includes methods for setting up the server, handling client connections, and communicating with the I2C hub.

Author

Daan Breur

Definition in file wemosserver.cpp.

8.17.2 Macro Definition Documentation

8.17.2.1 BUFFER_SIZE

```
#define BUFFER_SIZE 1024
```

Maximum data size to be read or sent over the wire.

Definition at line 31 of file wemosserver.cpp.

8.17.2.2 MAX_CLIENTS

```
#define MAX_CLIENTS 128
```

Maximum number of clients that can be connected to the server.

Definition at line 36 of file wemosserver.cpp.

8.17.2.3 TAFEL_KNOP_1

```
#define TAFEL_KNOP_1 0x80
```

8.17.2.4 TAFEL_LAMP_1

```
#define TAFEL_LAMP_1 0x6D
```

8.18 wemosserver.cpp

Go to the documentation of this file.

```
00001
00011 #include "wemosserver.h"
00012
00013 #include <arpa/inet.h>
00014 #include <asm-generic/socket.h>
00015 #include <netinet/in.h>
00016 #include <string.h>
00017 #include <sys/socket.h>
00018 #include <unistd.h>
00019
00020 #include <iostream>
00021 #include <stdexcept>
00022 #include <string>
00023 #include <thread>
00024
00025 #include "packets.h"
00026 #include "slavemanager.h"
00027
00031 #define BUFFER_SIZE 1024
00032
00036 #define MAX_CLIENTS 128
00037
00038 // private methods start here
00039 void WemosServer::handleClient(int client_fd, const struct sockaddr_in client_address) {
00040     uint8_t buffer[BUFFER_SIZE] = {0};
00041     ssize_t bytes_received;
00042
00043     std::cout « "Thread " « std::this_thread::get_id() « " : Connection accepted from "
00044               « inet_ntoa(client_address.sin_addr) « ':' « ntohs(client_address.sin_port)
00045               « std::endl;
00046
00047     while ((bytes_received = recv(client_fd, buffer, BUFFER_SIZE, 0)) > 0) {
00048         printf("Received %zd bytes from %s:%d:\n", bytes_received,
00049                inet_ntoa(client_address.sin_addr), ntohs(client_address.sin_port));
00050
00051         for (int i = 0; i < bytes_received; i++) printf("%02X ", buffer[i]);
00052         printf("\n");
00053
00054         size_t offset = 0;
00055         while (offset + 2 <= bytes_received) {
00056             struct sensor_packet *pkt_ptr = (struct sensor_packet *)&buffer[offset];
00057             uint8_t data_length = pkt_ptr->header.length;
00058             PacketType ptype = pkt_ptr->header.ptype;
00059             SensorType s_type = pkt_ptr->data.generic.metadata.sensor_type;
00060             uint8_t s_id = pkt_ptr->data.generic.metadata.sensor_id;
00061
00062             if (offset + data_length + sizeof(struct sensor_header) > bytes_received) {
00063                 printf("Incomplete packet received, discarding\n");
00064                 break;
00065             }
00066
00067             switch (ptype) {
00068                 case PacketType::DATA:
00069                     printf("Packet length: %u, type: %u\n", data_length, s_type);
00070
00071                     processSensorData((const struct sensor_packet *)&buffer[offset]);
00072                     break;
00073
00074                 case PacketType::HEARTBEAT:
00075                     printf("Heartbeat packet: ID=%u, type=%u\n",
00076                            pkt_ptr->data.heartbeat.metadata.sensor_id,
00077                            pkt_ptr->data.heartbeat.metadata.sensor_type);
00078
00079                     // Register the slave device
00080                     slave_manager.registerSlave(pkt_ptr->data.heartbeat.metadata.sensor_id,
00081                                                 client_fd);
00082                     break;
00083
00084                 case PacketType::DASHBOARD_GET:
00085                     printf("Dashboard requested data on sensor: ID=%u, type=%u\n",
00086                            pkt_ptr->data.generic.metadata.sensor_id,
00087                            pkt_ptr->data.generic.metadata.sensor_type);
00088
00089                     if (s_id > 127) {
00090                         // YIPEE
00091                         struct sensor_packet s_packet =
00092                             slave_manager.getSlaveState(pkt_ptr->data.generic.metadata.sensor_id);
00093                         sendToDashboard(client_fd, &s_packet,
00094                                         sizeof(s_packet.header) + s_packet.header.length);
00095                     } else {
00096                         i2c_client.sendRawData((uint8_t *)pkt_ptr,
00097                                                sizeof(struct sensor_header) + data_length);
```

```
00098
00099                       printf("incoming data: ");
00100                       for (int i = 0; i < sizeof(struct sensor_header) + pkt_ptr->header.length;
00101                            ++i) {
00102                           printf("%02X ", ((uint8_t *)(pkt_ptr))[i]);
00103                       }
00104                       printf("\n");
00105                       struct sensor_packet ret_pkt;
00106                       do {
00107                           ret_pkt = i2c_client.retrievePacket(true);
00108                       } while (ret_pkt.data.generic.metadata.sensor_id !=
00109                                pkt_ptr->data.generic.metadata.sensor_id);
00110
00111                       printf("sending back to dashboard :D\n");
00112                       sendToDashboard(client_fd, pkt_ptr,
00113                                       sizeof(struct sensor_header) + data_length);
00114                   }
00115                   break;
00116
00117               case PacketType::DASHBOARD_POST:
00118                   printf("Dashboard posting data on sensor: ID=%u, type=%u\n",
00119                          pkt_ptr->data.generic.metadata.sensor_id,
00120                          pkt_ptr->data.generic.metadata.sensor_type);
00121                   // the dashboard is trying to update something
00122                   if (s_id > 127) {
00123                       // blabla
00124                       slave_manager.sendToSlave(
00125                           pkt_ptr->data.generic.metadata.sensor_id, (uint8_t *)pkt_ptr,
00126                           sizeof(struct sensor_header) + pkt_ptr->header.length);
00127                       slave_manager.updateSlaveState(pkt_ptr->data.generic.metadata.sensor_id,
00128                                                      *pkt_ptr);
00129                   } else {
00130                       i2c_client.sendRawData((uint8_t *)pkt_ptr,
00131                                              sizeof(struct sensor_header) + data_length);
00132                   }
00133                   break;
00134
00135               default:
00136                   // unknown packet type
00137                   break;
00138           }
00139
00140           offset += data_length + sizeof(struct sensor_header);
00141       }
00142   }
00143
00144   if (bytes_received == 0) {
00145       printf("Connection closed by %s:%d\n", inet_ntoa(client_address.sin_addr),
00146              ntohs(client_address.sin_port));
00147   } else if (bytes_received < 0) {
00148       perror("recv failed");
00149   }
00150
00151   close(client_fd);
00152 }
00153
00154 void WemosServer::processSensorData(const struct sensor_packet *packet) {
00155   uint8_t slave_id = packet->data.generic.metadata.sensor_id;
00156     slave_manager.updateSlaveState(slave_id, *packet);
00157
00158     #define TAFEL_KNOP_1 0x80
00159     #define TAFEL_LAMP_1 0x6D
00160
00161     switch (packet->data.generic.metadata.sensor_type) {
00162         case SensorType::BUTTON: {
00163             printf("Processing button data: ID=%u\n", slave_id);
00164
00165             printf("%hhu\n", TAFEL_KNOP_1);
00166             // TODO: een tafel-knop ingedrukt
00167             switch (slave_id) {
00168                 case TAFEL_KNOP_1:
00169                     // tafel knop 1
00170                     struct sensor_packet pkt = {.header = {.length = sizeof(struct
      sensor_packet_light), .ptype = PacketType::DATA}, .data = {.light = {.metadata = {.sensor_type =
      SensorType::LIGHT, .sensor_id = TAFEL_LAMP_1}}}};
00171
00172                     struct sensor_packet led_state;
00173                     led_state.header.ptype = PacketType::DASHBOARD_GET;
00174                     led_state.header.length = sizeof(struct sensor_packet_light);
00175                     led_state.data.light.metadata.sensor_id = TAFEL_LAMP_1;
00176                     led_state.data.light.metadata.sensor_type = SensorType::LIGHT;
00177
00178                     i2c_client.sendRawData((uint8_t*)&led_state, sizeof(struct sensor_header) +
      led_state.header.length);
00179                     led_state.data.light.target_state =
      !i2c_client.retrievePacket(true).data.light.target_state;
00180                     printf("led state = %hhu\n", led_state.data.light.target_state);
```

```
00181
00182                     led_state.header.ptype = PacketType::DASHBOARD_POST;
00183                     i2c_client.sendRawData((uint8_t*)&led_state, sizeof(struct sensor_header) +
      led_state.header.length);
00184
00185                 break;
00186               }
00187             break;
00188         }
00189
00190         default:
00191             printf("No action defined for sensor type %u\n",
00192                    packet->data.generic.metadata.sensor_type);
00193             break;
00194     }
00195 }
00196
00197 void WemosServer::sendToDashboard(int dashboard_fd, struct sensor_packet *pkt_ptr, size_t len) {
00198     send(dashboard_fd, pkt_ptr, len, 0);
00199 }
00200 // private methods end here
00201
00202 WemosServer::WemosServer(int port, const std::string &hub_ip, int hub_port)
00203     : server_fd(-1), hub_ip(hub_ip), hub_port(hub_port), i2c_client() {
00204     if (port <= 0 || port > 65535) throw std::invalid_argument("Invalid listen port number");
00205
00206     if (INADDR_NONE == inet_addr(hub_ip.c_str()))
00207         throw std::invalid_argument("Invalid hub IP address passed");
00208     if (hub_port <= 0 || hub_port > 65535) throw std::invalid_argument("Invalid hub port passed");
00209
00210     memset(&listen_address, 0, sizeof(listen_address));
00211     listen_address.sin_family = AF_INET;
00212     listen_address.sin_addr = {INADDR_ANY};
00213     listen_address.sin_port = htons(port);
00214 }
00215
00216 WemosServer::~WemosServer() {
00217     tearDown();
00218     // other shit
00219 }
00220
00221 void WemosServer::socketSetup() {
00222     if ((server_fd = socket(AF_INET, SOCK_STREAM | SOCK_NONBLOCK, 0)) < 0) {
00223         perror("socket() failed");
00224         throw std::runtime_error("socket() failed");
00225         exit(EXIT_FAILURE);
00226     }
00227
00228     const int enable_opt = 1;
00229     if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &enable_opt,
00230                    sizeof(enable_opt)) < 0) {
00231         perror("setsockopt() failed");
00232         throw std::runtime_error("setsockopt() failed");
00233         exit(EXIT_FAILURE);
00234     }
00235
00236     if (bind(server_fd, (struct sockaddr *)&listen_address, sizeof(listen_address)) < 0) {
00237         perror("bind() failed");
00238         throw std::runtime_error("bind() failed");
00239         exit(EXIT_FAILURE);
00240     }
00241
00242     if (listen(server_fd, MAX_CLIENTS) < 0) {
00243         perror("listen() failed");
00244         throw std::runtime_error("listen() failed");
00245         exit(EXIT_FAILURE);
00246     }
00247
00248     std::cout « "Listening on port " « ntohs(listen_address.sin_port) « " (max " « MAX_CLIENTS
00249               « " clients)" « std::endl;
00250 }
00251
00252 void WemosServer::setupI2cClient() { i2c_client.setup(hub_ip, hub_port); }
00253
00254 void WemosServer::start() {
00255     socketSetup();
00256
00257     setupI2cClient();
00258     i2c_client.openConnection();
00259     i2c_client.start();
00260
00261     while (true) {
00262         struct sensor_packet pkt;
00263         try {
00264             // pkt = i2c_client.retrievePacket();
00265
00266             // std::cout « "packet received from the I2C hub!" « std::endl;
```

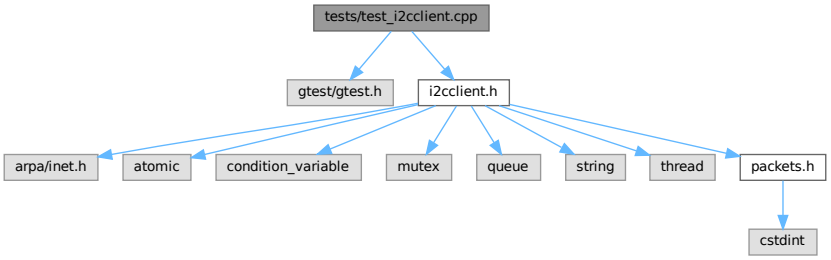
```
00267
00268             // now do things with the I2C packet if necessary
00269
00270         } catch (std::runtime_error &exc) {
00271             /* this means there is no new I2C packet available */
00272             // std::cerr « exc.what() « std::endl;
00273         }
00274
00275         struct sockaddr_in client_address;
00276         socklen_t client_addr_len = sizeof(client_address);
00277         int client_fd = accept(server_fd, (struct sockaddr *)&client_address, &client_addr_len);
00278
00279         if (-1 == client_fd) {
00280             // no one tried to connect
00281             continue;
00282         }
00283
00284         std::cout « "Connection accepted from " « inet_ntoa(client_address.sin_addr) « ":"
00285                   « ntohs(client_address.sin_port) « std::endl;
00286
00287         new std::thread(&WemosServer::handleClient, this, client_fd, client_address);
00288         // printf("aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\n");
00289     }
00290 }
00291
00292 void WemosServer::tearDown() {
00293     close(server_fd);
00294     i2c_client.closeConnection();
00295 }
```

8.19 tests/test_i2cclient.cpp File Reference

Unit tests for I2CClient class.

```
#include <gtest/gtest.h>
#include "i2cclient.h"
```

Include dependency graph for test_i2cclient.cpp:



Functions

- TEST (I2CClientTests, setup_ValidPort)

  Test the setup() function with valid port numbers.
- TEST (I2CClientTests, setup_InvalidPort_Negative)
- TEST (I2CClientTests, setup_InvalidPort_Zero)
- TEST (I2CClientTests, setup_InvalidPort_High)

8.19.1 Detailed Description

Unit tests for I2CClient class.

Author

Daan Breur

Definition in file test_i2cclient.cpp.

8.20 test_i2cclient.cpp

Go to the documentation of this file.

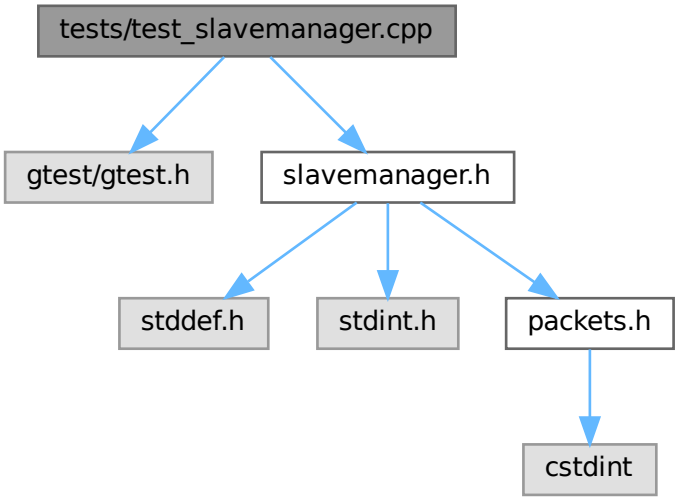
```
00001
00006 #include <gtest/gtest.h>
00007
00008 #include "i2cclient.h"
00009
00018 TEST(I2CClientTests, setup_ValidPort) {
00019     I2CClient server;
00020     EXPECT_NO_THROW(server.setup("10.0.0.1", 5000));
00021     EXPECT_NO_THROW(server.setup("10.0.0.1", 6969));
00022     EXPECT_NO_THROW(server.setup("10.0.0.1", 65535));
00023 }
00024
00032 TEST(I2CClientTests, setup_InvalidPort_Negative) {
00033     I2CClient server;
00034     EXPECT_THROW(server.setup("10.0.0.1", -1), std::invalid_argument);
00035 }
00036
00044 TEST(I2CClientTests, setup_InvalidPort_Zero) {
00045     I2CClient server;
00046     EXPECT_THROW(server.setup("10.0.0.1", 0), std::invalid_argument);
00047 }
00048
00057 TEST(I2CClientTests, setup_InvalidPort_High) {
00058     I2CClient server;
00059     EXPECT_THROW(server.setup("10.0.0.1", 65536), std::invalid_argument);
00060     EXPECT_THROW(server.setup("10.0.0.1", 69696), std::invalid_argument);
00061 }
```

8.21 tests/test_slavemanager.cpp File Reference

Unit tests for SlaveManager class.

```
#include <gtest/gtest.h>
#include "slavemanager.h"
```

Include dependency graph for test_slavemanager.cpp:



Functions

- TEST (SlaveManagerTests, RegisterSlave)
- TEST (SlaveManagerTests, UnregisterSlave)

8.21.1 Detailed Description

Unit tests for SlaveManager class.

Author

Daan Breur

Definition in file test_slavemanager.cpp.

8.21.2 Function Documentation

8.21.2.1 TEST() [1/2]

```
TEST (
            SlaveManagerTests ,
            RegisterSlave  )
```

Definition at line 10 of file test_slavemanager.cpp.

8.21.2.2 TEST() [2/2]

```
TEST (
            SlaveManagerTests ,
            UnregisterSlave  )
```

Definition at line 18 of file test_slavemanager.cpp.

8.22 test_slavemanager.cpp

Go to the documentation of this file.

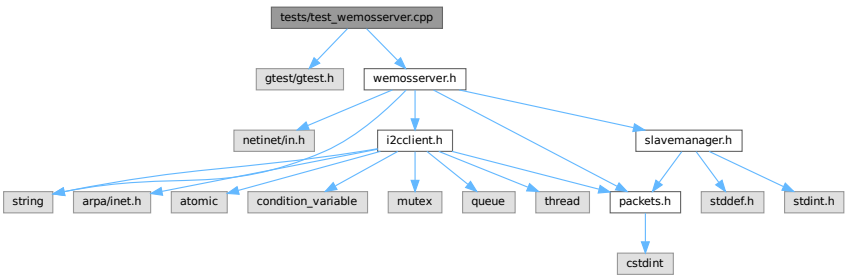
```
00001
00006 #include <gtest/gtest.h>
00007
00008 #include "slavemanager.h"
00009
00010 TEST(SlaveManagerTests, RegisterSlave) {
00011     SlaveManager manager;
00012     int fd = 5;
00013
00014     EXPECT_NO_THROW(manager.registerSlave(1, fd));
00015     EXPECT_EQ(manager.getSlaveFD(1), fd);
00016 }
00017
00018 TEST(SlaveManagerTests, UnregisterSlave) {
00019     SlaveManager manager;
00020     int fd = 5;
00021
00022     EXPECT_NO_THROW(manager.registerSlave(1, fd));
00023     EXPECT_EQ(manager.getSlaveFD(1), fd);
00024
00025     EXPECT_NO_THROW(manager.unregisterSlave(1));
00026     EXPECT_EQ(manager.getSlaveFD(1), -1);
00027 }
```

8.23 tests/test_wemosserver.cpp File Reference

Unit tests for WemosServer class.

```
#include <gtest/gtest.h>
#include "wemosserver.h"
```

Include dependency graph for test_wemosserver.cpp:



Functions

- TEST (WemosServerTest, Constructor_ValidPort)

  Test the constructor with valid port numbers.
- TEST (WemosServerTest, Constructor_InvalidPort_Negative)
- TEST (WemosServerTest, Constructor_InvalidPort_Zero)
- TEST (WemosServerTest, Constructor_InvalidPort_High)
- TEST (WemosServerTest, Constructor_ValidHubIPAddress)
- TEST (WemosServerTest, Constructor_InvalidHubIPAddress)

  Test the constructor with invalid hub IP addresses.
- TEST (WemosServerTest, Constructor_ValidHubPort)

  Test the constructor with valid hub port numbers.
- TEST (WemosServerTest, Constructor_InvalidHubPort_Negative)

  Test the constructor with invalid negative hub port numbers.
- TEST (WemosServerTest, Constructor_InvalidHubPort_High)

  Test the constructor with invalid hub port numbers.
- TEST (WemosServerTest, Constructor_InvalidHubPort_Zero)

  Test the constructor with invalid hub port numbers.

8.23.1 Detailed Description

Unit tests for WemosServer class.

This file contains unit tests for the WemosServer class, which handles the server functionality for the Wemos device. The tests cover various aspects of the class, including socket setup, client handling, and communication with the I2C hub.

Author

Daan Breur

Definition in file test_wemosserver.cpp.

8.23.2 Function Documentation

8.23.2.1 TEST() [1/10]

```
TEST (
            WemosServerTest ,
            Constructor_InvalidHubIPAddress  )
```

Test the constructor with invalid hub IP addresses.

Test WemosServerTest.Constructor_InvalidHubIPAddress

- Verify that the constructor throws an exception when an invalid hub IP address is provided.
- Testcases include:
  - Empty string
  - Invalid IP format (e.g., "192.168.0")
  - Non-numeric characters (e.g., "invalid_ip")
  - Out of range IP address (e.g., "256.256.256.256")
- Expects std::invalid_argument to be thrown.

Definition at line 88 of file test_wemosserver.cpp.

8.23.2.2 TEST() [2/10]

```
TEST (
            WemosServerTest ,
            Constructor_InvalidHubPort_High  )
```

Test the constructor with invalid hub port numbers.

Test WemosServerTest.Constructor_InvalidHubPort_High

- Test the constructor of WemosServer with invalid hub port numbers that are to high.
- Expect std::invalid_argument to be thrown.

Definition at line 125 of file test_wemosserver.cpp.

8.23.2.3 TEST() [3/10]

```
TEST (
            WemosServerTest ,
            Constructor_InvalidHubPort_Negative  )
```

Test the constructor with invalid negative hub port numbers.

Test WemosServerTest.Constructor_InvalidHubPort_Negative

- Test the constructor of WemosServer with invalid hub port numbers that are negative.
- Expect std::invalid_argument to be thrown.

Definition at line 113 of file test_wemosserver.cpp.

8.23.2.4 TEST() [4/10]

```
TEST (
            WemosServerTest ,
            Constructor_InvalidHubPort_Zero  )
```

Test the constructor with invalid hub port numbers.

Test WemosServerTest.Constructror_InvalidHubPort_Zero

- Test the constructor of WemosServer with invalid hub port numbers that are zero.
- Expect std::invalid_argument to be thrown.

Definition at line 138 of file test_wemosserver.cpp.

8.23.2.5 TEST() [5/10]

```
TEST (
            WemosServerTest ,
            Constructor_InvalidPort_High  )
```

Test WemosServerTest.Constructor_InvalidPort_High

- Verify that the constructor throws an exception when a port number greater than 65535 is provided.
- Expects std::invalid_argument to be thrown.

Definition at line 58 of file test_wemosserver.cpp.

8.23.2.6 TEST() [6/10]

```
TEST (
            WemosServerTest ,
            Constructor_InvalidPort_Negative  )
```

Test WemosServerTest.Constructor_InvalidPort_Negative

- Verify that the constructor throws an exception when a negative port number is provided.
- Expects std::invalid_argument to be thrown.

Definition at line 35 of file test_wemosserver.cpp.

8.23.2.7 TEST() [7/10]

```
TEST (
            WemosServerTest ,
            Constructor_InvalidPort_Zero  )
```

Test WemosServerTest.Constructor_InvalidPort_Zero

- Verify that the constructor throws an exception when a port number of zero is provided.
- Expects std::invalid_argument to be thrown.

Definition at line 46 of file test_wemosserver.cpp.

8.23.2.8 TEST() [8/10]

```
TEST (
            WemosServerTest ,
            Constructor_ValidHubIPAddress  )
```

Test WemosServerTest.Constructor_ValidHubIPAddress

- Test the constructor of WemosServer with valid hub IP addresses.
- Expect no exceptions to be thrown.

Definition at line 70 of file test_wemosserver.cpp.

8.23.2.9 TEST() [9/10]

```
TEST (
            WemosServerTest ,
            Constructor_ValidHubPort  )
```

Test the constructor with valid hub port numbers.

Test WemosServerTest.Constructor_ValidHubPort

Definition at line 99 of file test_wemosserver.cpp.

8.23.2.10 TEST() [10/10]

```
TEST (
            WemosServerTest ,
            Constructor_ValidPort  )
```

Test the constructor with valid port numbers.

Test WemosServerTest.Constructor_ValidPort

- Test the constructor of WemosServer with valid port numbers.
- Expect no exceptions to be thrown.

Definition at line 22 of file test_wemosserver.cpp.

8.24 test_wemosserver.cpp

Go to the documentation of this file.

```
00001
00010 #include <gtest/gtest.h>
00011
00012 #include "wemosserver.h"
00013
00022 TEST(WemosServerTest, Constructor_ValidPort) {
00023     EXPECT_NO_THROW(WemosServer server(5000, "10.0.0.1", 5000));
00024     EXPECT_NO_THROW(WemosServer server(6969, "10.0.0.1", 5000));
00025     EXPECT_NO_THROW(WemosServer server(65535, "10.0.0.1", 5000));
00026 }
00027
00035 TEST(WemosServerTest, Constructor_InvalidPort_Negative) {
00036     EXPECT_THROW(WemosServer server(-1, "10.0.0.1", 5000), std::invalid_argument);
00037 }
00038
00046 TEST(WemosServerTest, Constructor_InvalidPort_Zero) {
00047     EXPECT_THROW(WemosServer server(0, "10.0.0.1", 5000), std::invalid_argument);
00048 }
00049
00058 TEST(WemosServerTest, Constructor_InvalidPort_High) {
00059     EXPECT_THROW(WemosServer server(65536, "10.0.0.1", 5000), std::invalid_argument);
00060     EXPECT_THROW(WemosServer server(69696, "10.0.0.1", 5000), std::invalid_argument);
00061 }
00062
00070 TEST(WemosServerTest, Constructor_ValidHubIPAddress) {
00071     EXPECT_NO_THROW(WemosServer server(5000, "10.0.0.1", 5000));
00072     EXPECT_NO_THROW(WemosServer server(5000, "192.168.10.10", 5000));
00073 }
00074
00088 TEST(WemosServerTest, Constructor_InvalidHubIPAddress) {
00089     EXPECT_THROW(WemosServer server(5000, "", 5000), std::invalid_argument);
00090     EXPECT_THROW(WemosServer server(5000, "invalid_ip", 5000), std::invalid_argument);
00091     EXPECT_THROW(WemosServer server(5000, "256.256.256.256", 5000), std::invalid_argument);
00092 }
00093
00099 TEST(WemosServerTest, Constructor_ValidHubPort) {
00100     EXPECT_NO_THROW(WemosServer server(5000, "10.0.0.1", 5000));
00101     EXPECT_NO_THROW(WemosServer server(5000, "10.0.0.1", 6969));
00102     EXPECT_NO_THROW(WemosServer server(5000, "10.0.0.1", 65535));
00103 }
00104
00113 TEST(WemosServerTest, Constructor_InvalidHubPort_Negative) {
00114     EXPECT_THROW(WemosServer server(5000, "10.0.0.1", -1), std::invalid_argument);
00115 }
00116
00125 TEST(WemosServerTest, Constructor_InvalidHubPort_High) {
00126     EXPECT_THROW(WemosServer server(5000, "10.0.0.1", 65536), std::invalid_argument);
00127     EXPECT_THROW(WemosServer server(5000, "10.0.0.1", 69696), std::invalid_argument);
00128 }
00129
00138 TEST(WemosServerTest, Constructor_InvalidHubPort_Zero) {
00139     EXPECT_THROW(WemosServer server(5000, "10.0.0.1", 0), std::invalid_argument);
00140 }
```

Index