

Wemos Bridge Server

commit-0692db8

Generated by Doxygen 1.9.8

1 Wemos Bridge Server	1
2 Test List	3
3 Topic Index	5
3.1 Topics	5
4 Class Index	7
4.1 Class List	7
5 File Index	9
5.1 File List	9
6 Topic Documentation	11
6.1 Tests	11
6.1.1 Detailed Description	11
6.1.2 WemosServerTests	12
6.1.3 I2CClientTests	12
6.1.3.1 Detailed Description	12
6.1.3.2 Function Documentation	13
6.1.4 SlaveManagerTests	14
6.2 Packets	14
6.2.1 Detailed Description	15
7 Class Documentation	17
7.1 I2CClient::DataReceiveReturn Struct Reference	17
7.1.1 Detailed Description	17
7.1.2 Member Data Documentation	18
7.1.2.1 data	18
7.1.2.2 length	18
7.2 I2CClient Class Reference	18
7.2.1 Detailed Description	19
7.2.2 Constructor & Destructor Documentation	19
7.2.2.1 I2CClient() [1/3]	19
7.2.2.2 ~I2CClient()	20
7.2.2.3 I2CClient() [2/3]	20
7.2.2.4 I2CClient() [3/3]	20
7.2.3 Member Function Documentation	20
7.2.3.1 closeConnection()	20
7.2.3.2 openConnection()	20
7.2.3.3 operator=() [1/2]	21
7.2.3.4 operator=() [2/2]	21
7.2.3.5 receiveLoop()	21
7.2.3.6 retrievePacket()	21

7.2.3.7 sendRawData()	22
7.2.3.8 setup()	22
7.2.3.9 start()	23
7.2.4 Member Data Documentation	23
7.2.4.1 client_fd	23
7.2.4.2 connected	23
7.2.4.3 hub_address	23
7.2.4.4 queue_condition	23
7.2.4.5 queue_mutex	23
7.2.4.6 read_packets_queue	24
7.2.4.7 receive_mutex	24
7.2.4.8 receive_thread	24
7.2.4.9 running	24
7.3 sensor_data Union Reference	24
7.3.1 Detailed Description	25
7.3.2 Member Data Documentation	25
7.3.2.1 co2	25
7.3.2.2 generic	25
7.3.2.3 heartbeat	25
7.3.2.4 humidity	25
7.3.2.5 light	25
7.3.2.6 rgb_light	26
7.3.2.7 temperature	26
7.4 sensor_packet::sensor_data Union Reference	26
7.4.1 Detailed Description	27
7.4.2 Member Data Documentation	27
7.4.2.1 co2	27
7.4.2.2 generic	27
7.4.2.3 heartbeat	27
7.4.2.4 humidity	27
7.4.2.5 light	27
7.4.2.6 rgb_light	27
7.4.2.7 temperature	28
7.5 sensor_header Struct Reference	28
7.5.1 Detailed Description	28
7.5.2 Member Data Documentation	29
7.5.2.1 length	29
7.5.2.2 ptype	29
7.6 sensor_heartbeat Struct Reference	29
7.6.1 Detailed Description	30
7.6.2 Member Data Documentation	30
7.6.2.1 metadata	30

7.7 sensor_metadata Struct Reference	30
7.7.1 Detailed Description	31
7.7.2 Member Data Documentation	31
7.7.2.1 sensor_id	31
7.7.2.2 sensor_type	31
7.8 sensor_packet Struct Reference	31
7.8.1 Detailed Description	32
7.8.2 Member Data Documentation	33
7.8.2.1 data	33
7.8.2.2 header	33
7.9 sensor_packet_co2 Struct Reference	33
7.9.1 Detailed Description	34
7.9.2 Member Data Documentation	34
7.9.2.1 metadata	34
7.9.2.2 value	34
7.10 sensor_packet_generic Struct Reference	35
7.10.1 Detailed Description	35
7.10.2 Member Data Documentation	36
7.10.2.1 metadata	36
7.11 sensor_packet_humidity Struct Reference	36
7.11.1 Detailed Description	37
7.11.2 Member Data Documentation	37
7.11.2.1 metadata	37
7.11.2.2 value	37
7.12 sensor_packet_light Struct Reference	37
7.12.1 Detailed Description	38
7.12.2 Member Data Documentation	38
7.12.2.1 metadata	38
7.12.2.2 target_state	39
7.13 sensor_packet_rgb_light Struct Reference	39
7.13.1 Detailed Description	40
7.13.2 Member Data Documentation	40
7.13.2.1 blue_state	40
7.13.2.2 green_state	40
7.13.2.3 metadata	40
7.13.2.4 red_state	40
7.14 sensor_packet_temperature Struct Reference	41
7.14.1 Detailed Description	41
7.14.2 Member Data Documentation	42
7.14.2.1 metadata	42
7.14.2.2 value	42
7.15 SlaveDevice Struct Reference	42

7.15.1 Detailed Description	43
7.15.2 Member Data Documentation	43
7.15.2.1 fd	43
7.15.2.2 sensor_data	43
7.16 SlaveManager Class Reference	43
7.16.1 Detailed Description	44
7.16.2 Constructor & Destructor Documentation	44
7.16.2.1 SlaveManager() [1/3]	44
7.16.2.2 ~SlaveManager()	44
7.16.2.3 SlaveManager() [2/3]	44
7.16.2.4 SlaveManager() [3/3]	45
7.16.3 Member Function Documentation	45
7.16.3.1 getSlaveDevice()	45
7.16.3.2 getSlaveFD()	45
7.16.3.3 operator=() [1/2]	45
7.16.3.4 operator=() [2/2]	46
7.16.3.5 registerSlave()	46
7.16.3.6 sendToSlave()	46
7.16.3.7 unregisterSlave()	47
7.16.4 Member Data Documentation	47
7.16.4.1 slave_devices	47
7.17 WemosServer Class Reference	47
7.17.1 Detailed Description	49
7.17.2 Constructor & Destructor Documentation	49
7.17.2.1 WemosServer() [1/3]	49
7.17.2.2 ~WemosServer()	49
7.17.2.3 WemosServer() [2/3]	50
7.17.2.4 WemosServer() [3/3]	50
7.17.3 Member Function Documentation	50
7.17.3.1 handleClient()	50
7.17.3.2 operator=() [1/2]	50
7.17.3.3 operator=() [2/2]	50
7.17.3.4 processSensorData()	50
7.17.3.5 sendToDashboard()	50
7.17.3.6 setupI2cClient()	51
7.17.3.7 socketSetup()	51
7.17.3.8 start()	51
7.17.3.9 tearDown()	51
7.17.4 Member Data Documentation	51
7.17.4.1 hub_ip	51
7.17.4.2 hub_port	52
7.17.4.3 i2c_client	52

7.17.4.4 listen_address	52
7.17.4.5 server_fd	52
7.17.4.6 slave_manager	52
8 File Documentation	53
8.1 include/i2cclient.h File Reference	53
8.1.1 Detailed Description	54
8.2 i2cclient.h	54
8.3 include/packets.h File Reference	55
8.3.1 Detailed Description	57
8.3.2 Enumeration Type Documentation	57
8.3.2.1 PacketType	57
8.3.2.2 SensorType	58
8.3.3 Function Documentation	58
8.3.3.1 __attribute__((.	58
8.3.4 Variable Documentation	58
8.3.4.1 blue_state	58
8.3.4.2 data	58
8.3.4.3 green_state	58
8.3.4.4 header	59
8.3.4.5 length	59
8.3.4.6 metadata	59
8.3.4.7 ptype	59
8.3.4.8 red_state	59
8.3.4.9 sensor_id	59
8.3.4.10 sensor_type	60
8.3.4.11 target_state	60
8.3.4.12 value	60
8.4 packets.h	60
8.5 include/slavemanager.h File Reference	61
8.5.1 Detailed Description	62
8.5.2 Macro Definition Documentation	63
8.5.2.1 MAX_SLAVE_ID	63
8.6 slavemanager.h	63
8.7 include/wemosserver.h File Reference	63
8.7.1 Detailed Description	64
8.8 wemosserver.h	65
8.9 modules.dox File Reference	65
8.10 README.md File Reference	65
8.11 src/i2cclient.cpp File Reference	65
8.11.1 Detailed Description	66
8.11.2 Macro Definition Documentation	66

8.11.2.1 BUFFER_SIZE	66
8.11.2.2 THREAD_RELINQUISH	66
8.12 i2cclient.cpp	67
8.13 src/main.cpp File Reference	69
8.13.1 Detailed Description	70
8.13.2 Macro Definition Documentation	70
8.13.2.1 I2C_HUB_IP	70
8.13.2.2 I2C_HUB_PORT	70
8.13.2.3 SERVER_PORT	71
8.13.3 Function Documentation	71
8.13.3.1 global_shutdown_flag()	71
8.13.3.2 main()	71
8.13.3.3 signalHandler()	71
8.14 main.cpp	71
8.15 src/slavemanager.cpp File Reference	72
8.15.1 Detailed Description	72
8.16 slavemanager.cpp	72
8.17 src/wemosserver.cpp File Reference	73
8.17.1 Detailed Description	74
8.17.2 Macro Definition Documentation	74
8.17.2.1 BUFFER_SIZE	74
8.17.2.2 MAX_CLIENTS	75
8.18 wemosserver.cpp	75
8.19 tests/test_i2cclient.cpp File Reference	78
8.19.1 Detailed Description	78
8.20 test_i2cclient.cpp	79
8.21 tests/test_slavemanager.cpp File Reference	79
8.21.1 Detailed Description	80
8.21.2 Function Documentation	80
8.21.2.1 TEST() [1/2]	80
8.21.2.2 TEST() [2/2]	80
8.22 test_slavemanager.cpp	80
8.23 tests/test_wemosserver.cpp File Reference	81
8.23.1 Detailed Description	81
8.23.2 Function Documentation	82
8.23.2.1 TEST() [1/10]	82
8.23.2.2 TEST() [2/10]	82
8.23.2.3 TEST() [3/10]	82
8.23.2.4 TEST() [4/10]	83
8.23.2.5 TEST() [5/10]	83
8.23.2.6 TEST() [6/10]	83
8.23.2.7 TEST() [7/10]	84

8.23.2.8 TEST() [8/10]	84
8.23.2.9 TEST() [9/10]	84
8.23.2.10 TEST() [10/10]	85
8.24 test_wemossserver.cpp	85
Index	87

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:

Tests	11
WemosServerTests	12
I2CClientTests	12
SlaveManagerTests	14
Packets	14

Chapter 4

Class Index

4.1 Class List

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

I2CClient::DataReceiveReturn	17
I2CClient	18
sensor_data	
24	
sensor_packet::sensor_data	
26	
sensor_header	
Header structure for sensor packets	28
sensor_heartbeat	
Structure for heartbeat packets	29
sensor_metadata	
Structure for sensor metadata, which is always included in any packet	30
sensor_packet	
Union structure for the entire sensor packet	31
sensor_packet_co2	
Structure for CO2 sensor packets	33
sensor_packet_generic	
Structure for generic sensor packets	35
sensor_packet_humidity	
Structure for humidity sensor packets	36
sensor_packet_light	
Structure for light sensor packets	37
sensor_packet_rgb_light	
Structure for RGB light sensor packets	39
sensor_packet_temperature	
Structure for temperature sensor packets	41
SlaveDevice	
Structure representing a slave device	42
SlaveManager	43
WemosServer	47

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

include/i2cclient.h	
Header file for i2cclient.cpp	53
include/packets.h	
Header file for packets.h	55
include/slavemanager.h	
Header file for slavemanager.cpp	61
include/wemosserver.h	
Header file for wemosserver.cpp	63
src/i2cclient.cpp	
Implementation of I2CClient class	65
src/main.cpp	
Main entrypoint for Wemos Bridge Server application	69
src/slavemanager.cpp	
Implementation of SlaveManager class	72
src/wemosserver.cpp	
Implementation of WemosServer class	73
tests/test_i2cclient.cpp	
Unit tests for I2CClient class	78
tests/test_slavemanager.cpp	
Unit tests for SlaveManager class	79
tests/test_wemosserver.cpp	
Unit tests for WemosServer class	81

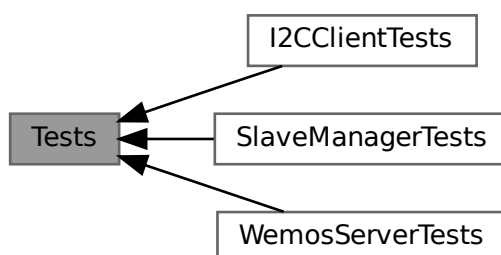
Chapter 6

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.

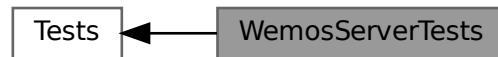
6.1.1 Detailed Description

Unit tests for the Wemos Bridge application.

6.1.2 WemosServerTests

All tests related to the [WemosServer](#) class.

Collaboration diagram for WemosServerTests:

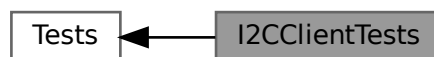


All tests related to the [WemosServer](#) class.

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

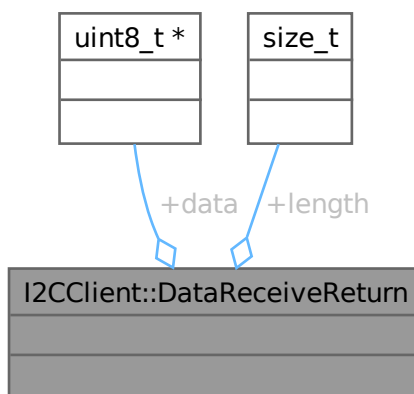
Chapter 7

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](#).

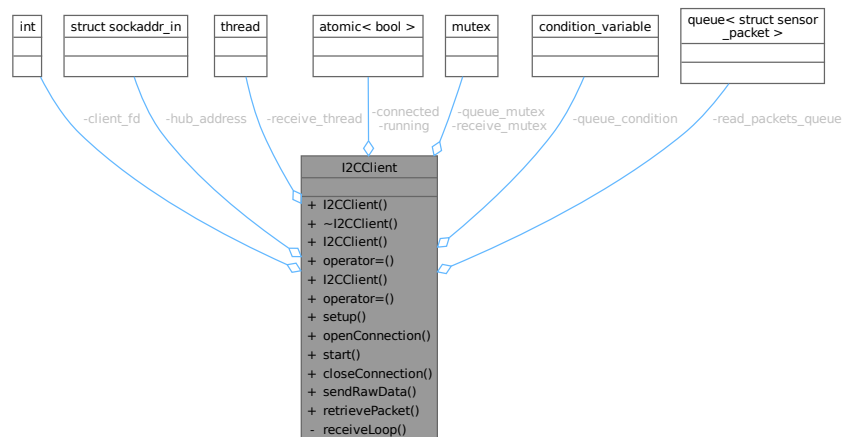
The documentation for this struct was generated from the following file:

- [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.

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

<code>std::invalid_argument</code>	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 193 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 149 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 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](#).

7.2.3.6 retrievePacket()

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

Sends packet data to the I2C hub.

Parameters

$t \leftrightarrow$	
<i>b.d.</i>	

Exceptions

<code>std::runtime_error</code>	if sending data fails.
---------------------------------	------------------------

Receives data from the I2C hub.

Parameters

<i>block</i>	Whether or not to block until a packet can be retrieved
--------------	---

Returns

A struct containing the received packet data.

Exceptions

<code>std::runtime_error</code>	if receiving data fails.
---------------------------------	--------------------------

Definition at line 212 of file [i2cclient.cpp](#).

7.2.3.7 sendRawData()

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

Internal method to send data to the I2C hub.

Parameters

<i>data</i>	The data to send to the I2C hub.
<i>length</i>	The length of the data to send.

Exceptions

<code>std::runtime_error</code>	if sending data fails.
---------------------------------	------------------------

Definition at line 205 of file [i2cclient.cpp](#).

7.2.3.8 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

<i>ip</i>	The IP address of the I2C hub.
<i>port</i>	The port number of the I2C hub.

Exceptions

<code>std::invalid_argument</code>	if an invalid IP address or port is passed
------------------------------------	--

Definition at line 137 of file [i2cclient.cpp](#).

7.2.3.9 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

<code>std::runtime_error</code>	if the client is not connected to the hub.
---------------------------------	--

Definition at line 181 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](#).

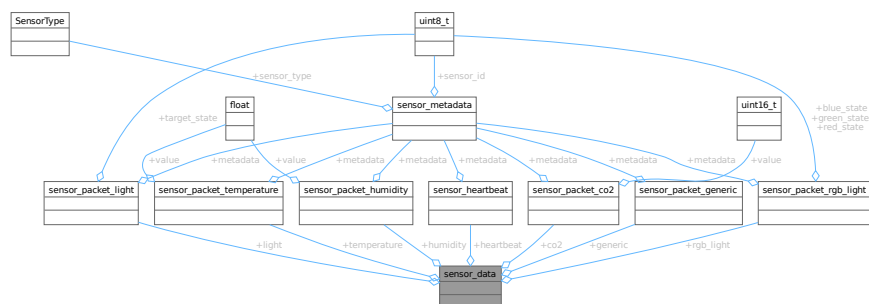
The documentation for this class was generated from the following files:

- [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

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 light

```
struct sensor\_packet\_light sensor_data::light
```

Definition at line 10 of file [packets.h](#).

7.3.2.6 rgb_light

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

Definition at line 11 of file [packets.h](#).

7.3.2.7 temperature

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

Definition at line 7 of file [packets.h](#).

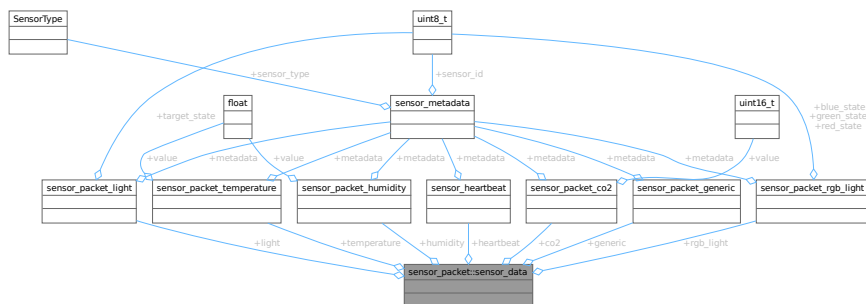
The documentation for this union was generated from the following file:

- [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

7.4.1 Detailed Description

Definition at line 227 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 231 of file [packets.h](#).

7.4.2.2 generic

```
struct sensor\_packet\_generic sensor_packet::sensor_data::generic
```

Definition at line 229 of file [packets.h](#).

7.4.2.3 heartbeat

```
struct sensor\_heartbeat sensor_packet::sensor_data::heartbeat
```

Definition at line 228 of file [packets.h](#).

7.4.2.4 humidity

```
struct sensor\_packet\_humidity sensor_packet::sensor_data::humidity
```

Definition at line 232 of file [packets.h](#).

7.4.2.5 light

```
struct sensor\_packet\_light sensor_packet::sensor_data::light
```

Definition at line 233 of file [packets.h](#).

7.4.2.6 rgb_light

```
struct sensor\_packet\_rgb\_light sensor_packet::sensor_data::rgb_light
```

Definition at line 234 of file [packets.h](#).

7.4.2.7 temperature

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

Definition at line 230 of file [packets.h](#).

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

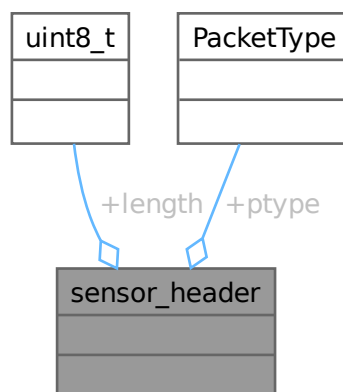
- [include/packets.h](#)

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 40 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 42 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 44 of file [packets.h](#).

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

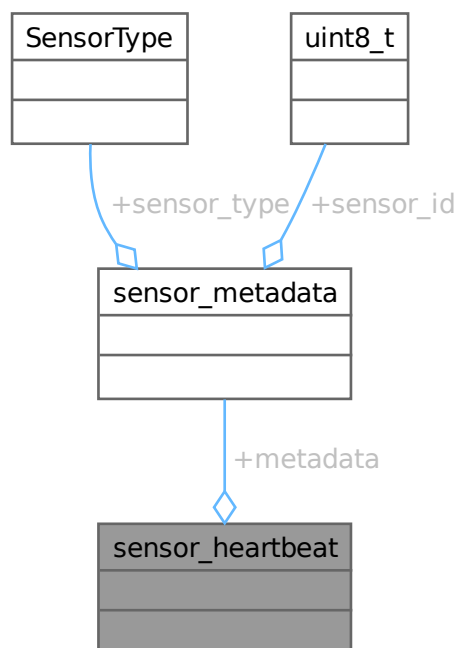
- [include/packets.h](#)

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 69 of file [packets.h](#).

7.6.2 Member Data Documentation

7.6.2.1 metadata

```
struct sensor\_metadata sensor_heartbeat::metadata
```

Definition at line 70 of file [packets.h](#).

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

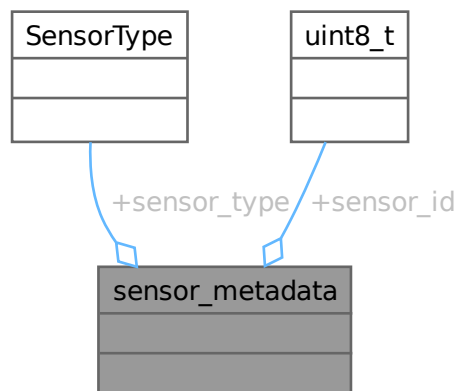
- [include/packets.h](#)

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 52 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 56 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 54 of file packets.h.

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

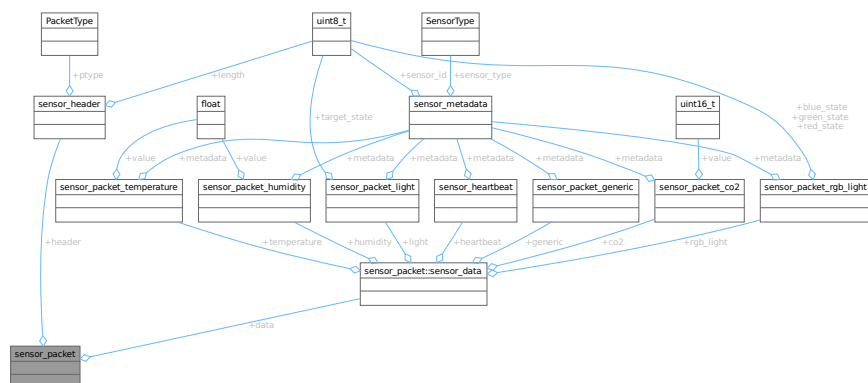
- `include/packets.h`

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 222 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 224 of file [packets.h](#).

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

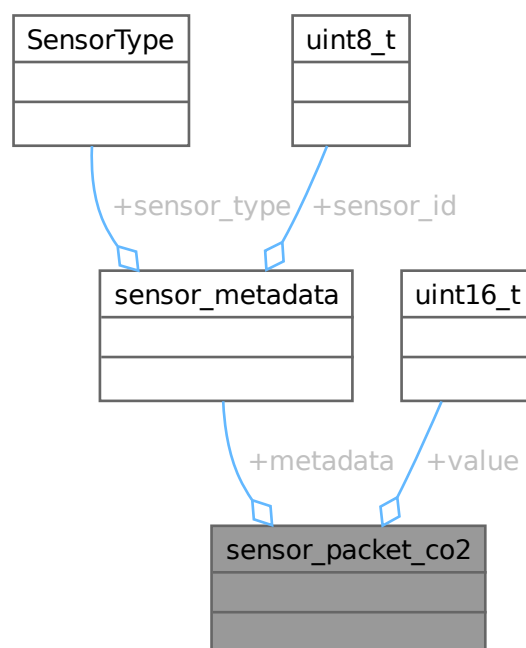
- [include/packets.h](#)

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 [107](#) 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 [108](#) 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 [110](#) of file [packets.h](#).

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

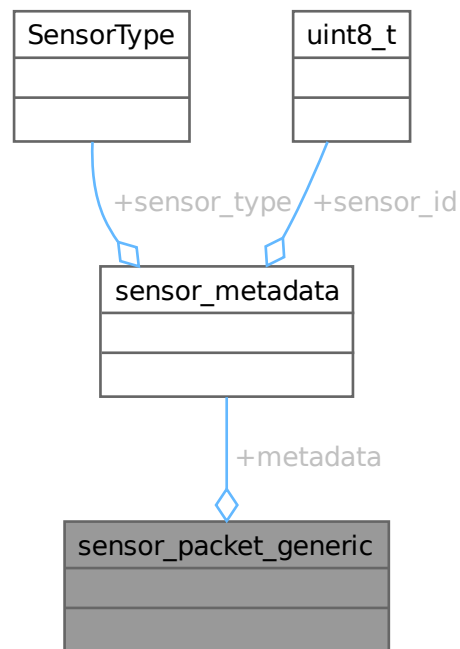
- [include/packets.h](#)

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 81 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 82 of file `packets.h`.

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

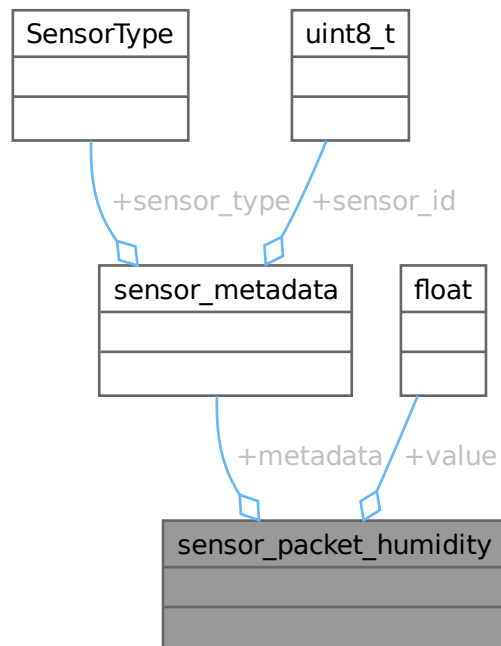
- `include/packets.h`

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 120 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 121 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 123 of file [packets.h](#).

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

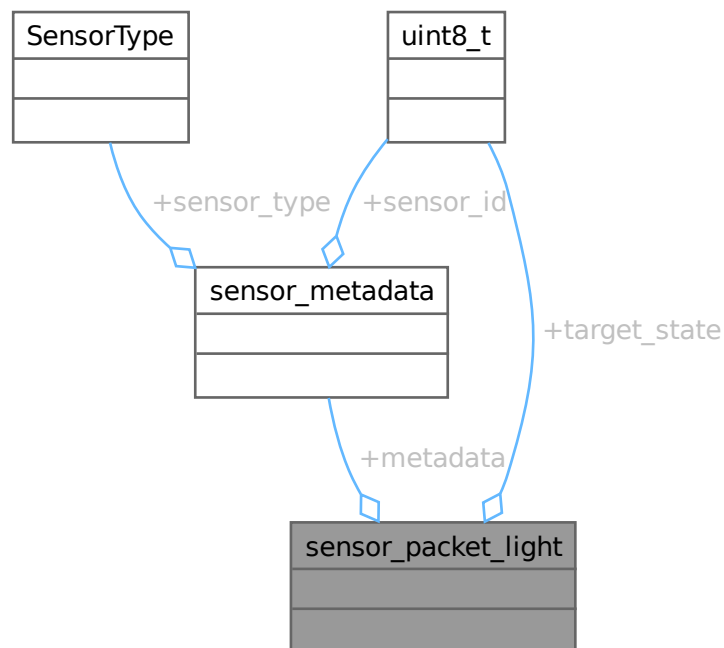
- [include/packets.h](#)

7.12 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.

7.12.1 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 133 of file `packets.h`.

7.12.2 Member Data Documentation

7.12.2.1 metadata

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

Definition at line 134 of file `packets.h`.

7.12.2.2 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 136 of file [packets.h](#).

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

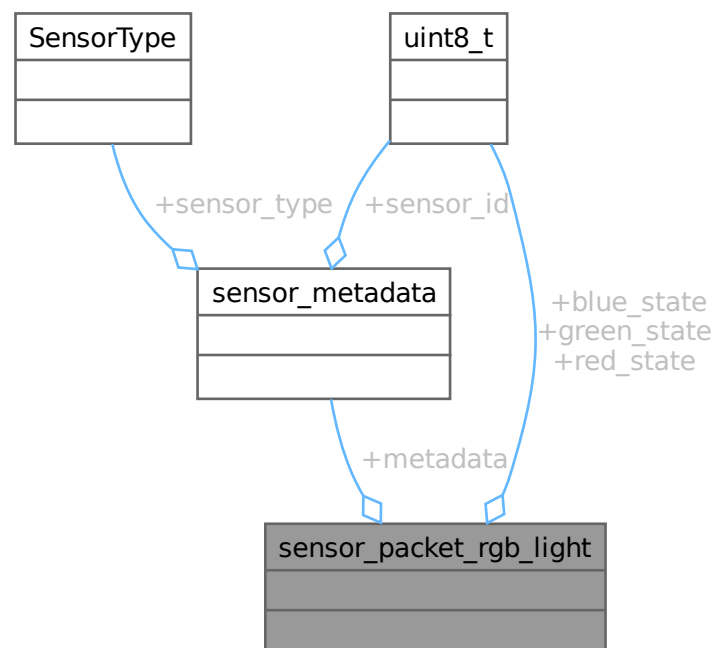
- [include/packets.h](#)

7.13 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.

7.13.1 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 147 of file [packets.h](#).

7.13.2 Member Data Documentation

7.13.2.1 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 154 of file [packets.h](#).

7.13.2.2 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 152 of file [packets.h](#).

7.13.2.3 metadata

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

Definition at line 148 of file [packets.h](#).

7.13.2.4 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 150 of file [packets.h](#).

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

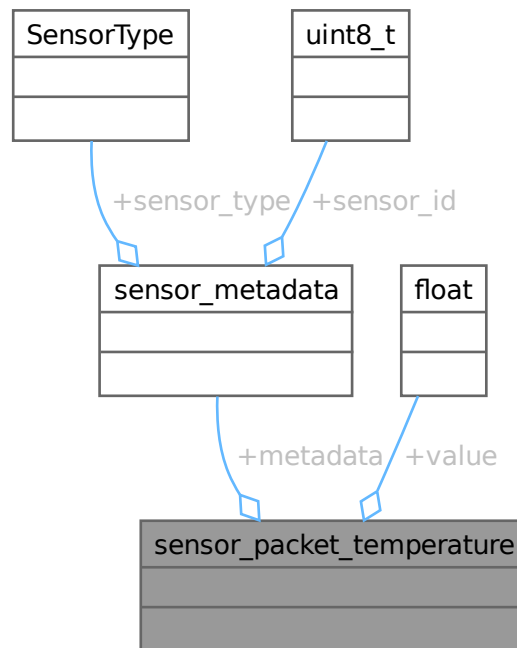
- [include/packets.h](#)

7.14 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.

7.14.1 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 94 of file [packets.h](#).

7.14.2 Member Data Documentation

7.14.2.1 metadata

```
struct sensor\_metadata sensor_packet_temperature::metadata
```

Definition at line 95 of file [packets.h](#).

7.14.2.2 value

```
float sensor_packet_temperature::value
```

Value of the sensor reading the temperature represented in Celcius.

Definition at line 97 of file [packets.h](#).

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

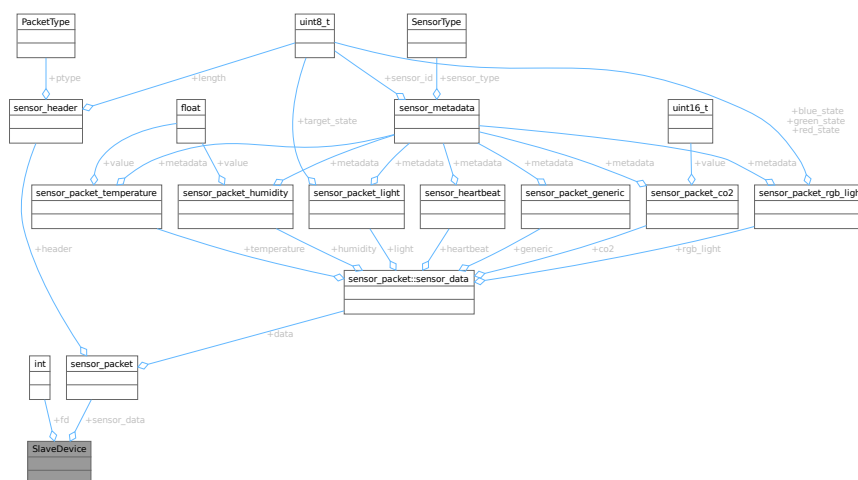
- [include/packets.h](#)

7.15 SlaveDevice Struct Reference

Structure representing a slave device.

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

Collaboration diagram for SlaveDevice:



Public Attributes

- `int` [fd](#)
- `struct` [sensor_packet](#) [sensor_data](#)

7.15.1 Detailed Description

Structure representing a slave device.

This structure contains the file descriptor associated with the slave device.

Definition at line 27 of file [slavemanager.h](#).

7.15.2 Member Data Documentation

7.15.2.1 fd

```
int SlaveDevice::fd
```

Definition at line 28 of file [slavemanager.h](#).

7.15.2.2 sensor_data

```
struct sensor_packet SlaveDevice::sensor_data
```

Definition at line 29 of file [slavemanager.h](#).

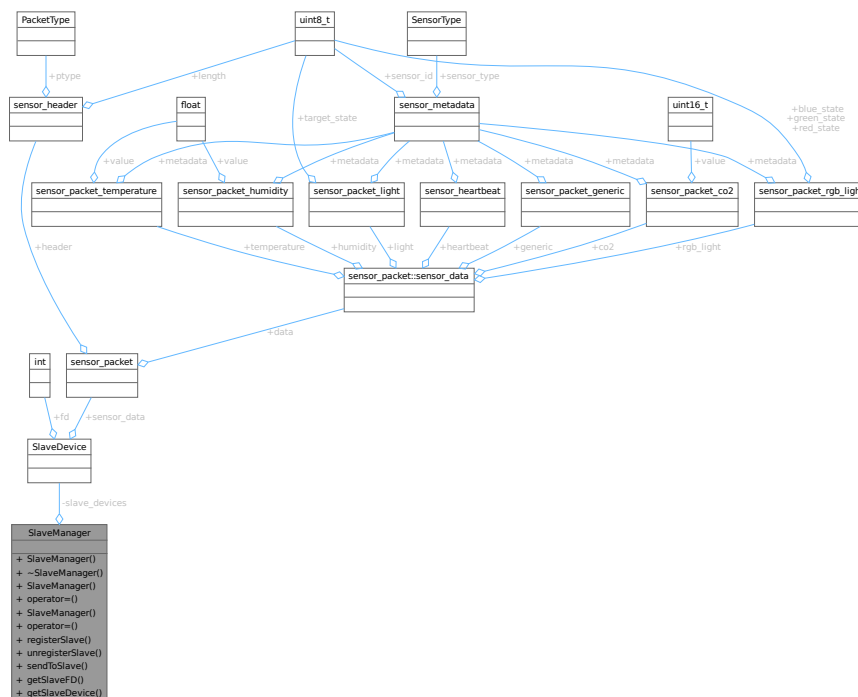
The documentation for this struct was generated from the following file:

- include/[slavemanager.h](#)

7.16 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.
- [SlaveDevice](#) [getSlaveDevice](#) (uint8_t slave_id) const
Gets the [SlaveDevice](#) associated with the given slave ID.

Private Attributes

- [SlaveDevice](#) slave_devices [MAX_SLAVE_ID+1]

7.16.1 Detailed Description

Definition at line 32 of file [slavemanager.h](#).

7.16.2 Constructor & Destructor Documentation

7.16.2.1 SlaveManager() [1/3]

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

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

7.16.2.2 ~SlaveManager()

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

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

7.16.2.3 SlaveManager() [2/3]

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

7.16.2.4 SlaveManager() [3/3]

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

7.16.3 Member Function Documentation

7.16.3.1 getSlaveDevice()

```
SlaveDevice SlaveManager::getSlaveDevice (
    uint8_t slave_id ) const
```

Gets the [SlaveDevice](#) associated with the given slave ID.

Parameters

<i>slave</i> _id	The ID of the slave device.
---------------------	-----------------------------

Returns

The [SlaveDevice](#) associated with the slave id.

Definition at line [84](#) of file [slavemanager.cpp](#).

7.16.3.2 getSlaveFD()

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

Gets the file descriptor associated with the given slave ID.

Parameters

<i>slave</i> _id	The ID of the slave device.
---------------------	-----------------------------

Returns

The file descriptor associated with the slave device.

Definition at line [75](#) of file [slavemanager.cpp](#).

7.16.3.3 operator=() [1/2]

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

7.16.3.4 operator=() [2/2]

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

7.16.3.5 registerSlave()

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

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

Parameters

<i>slave_id</i>	The ID of the slave device to register.
<i>fd</i>	The file descriptor associated with the slave device.

Exceptions

<i>std::invalid_argument</i>	if the slave ID is invalid.
------------------------------	-----------------------------

Definition at line 32 of file [slavemanager.cpp](#).

7.16.3.6 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

<i>slave_id</i>	The ID of the slave device to send data to.
<i>data</i>	The data to send to the slave device.
<i>length</i>	The length of the data to send.

Returns

0 on success, -1 on failure.

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

7.16.3.7 unregisterSlave()

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

Unregisters a slave device with the given ID.

Parameters

<code>slave_id</code>	The ID of the slave device to unregister.
-----------------------	---

Exceptions

<code>std::invalid_argument</code>	if the slave ID is invalid.
------------------------------------	-----------------------------

Warning

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

Definition at line 43 of file [slavemanager.cpp](#).

7.16.4 Member Data Documentation

7.16.4.1 slave_devices

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

Definition at line 34 of file [slavemanager.h](#).

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

- [include/slavemanager.h](#)
- [src/slavemanager.cpp](#)

7.17 WemosServer Class Reference

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


Private Attributes

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

7.17.1 Detailed Description

Definition at line 20 of file [wemosserver.h](#).

7.17.2 Constructor & Destructor Documentation**7.17.2.1 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

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

Exceptions

<code>std::invalid_argument</code>	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 187 of file [wemosserver.cpp](#).

7.17.2.2 ~WemosServer()

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

Definition at line 201 of file [wemosserver.cpp](#).

7.17.2.3 WemosServer() [2/3]

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

7.17.2.4 WemosServer() [3/3]

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

7.17.3 Member Function Documentation

7.17.3.1 handleClient()

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

Definition at line 39 of file [wemosserver.cpp](#).

7.17.3.2 operator=() [1/2]

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

7.17.3.3 operator=() [2/2]

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

7.17.3.4 processSensorData()

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

Definition at line 115 of file [wemosserver.cpp](#).

7.17.3.5 sendToDashboard()

```
void WemosServer::sendToDashboard (
    int dashboard_fd,
    uint8_t sensor_id ) [private]
```

Definition at line 175 of file [wemosserver.cpp](#).

7.17.3.6 setupI2cClient()

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

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

Definition at line 237 of file [wemosserver.cpp](#).

7.17.3.7 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

<code>std::runtime_error</code>	if socket creation, binding, or listening fails.
---------------------------------	--

Warning

This method should be called before starting the server loop.

Definition at line 206 of file [wemosserver.cpp](#).

7.17.3.8 start()

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

Definition at line 239 of file [wemosserver.cpp](#).

7.17.3.9 tearDown()

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

Definition at line 275 of file [wemosserver.cpp](#).

7.17.4 Member Data Documentation

7.17.4.1 hub_ip

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

Definition at line 26 of file [wemosserver.h](#).

7.17.4.2 hub_port

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

Definition at line 27 of file [wemosserver.h](#).

7.17.4.3 i2c_client

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

Definition at line 25 of file [wemosserver.h](#).

7.17.4.4 listen_address

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

Definition at line 23 of file [wemosserver.h](#).

7.17.4.5 server_fd

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

Definition at line 22 of file [wemosserver.h](#).

7.17.4.6 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

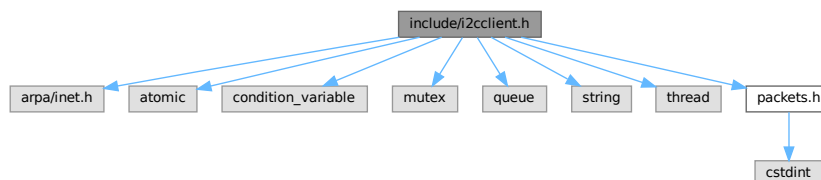
File Documentation

8.1 include/i2cclient.h File Reference

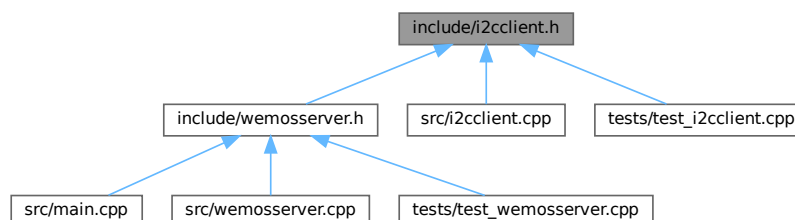
Header file for [i2cclient.cpp](#).

```
#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 I2CCCLIENT_H
00011 #define I2CCCLIENT_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
00129 #endif

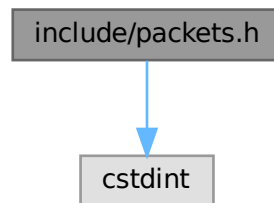
```

8.3 include/packets.h File Reference

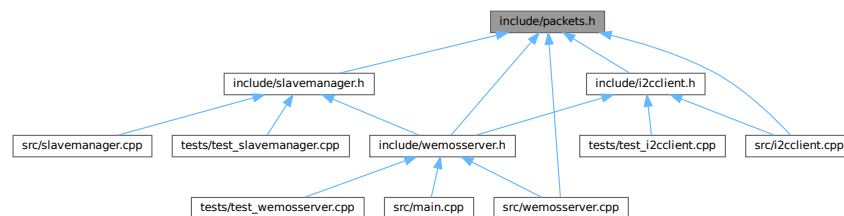
Header file for [packets.h](#).

```
#include <stdint>
```

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](#)
 - *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 }
- 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](#)

- `uint8_t target_state`
Value of the sensor reading the temperature represented in Celcius.
- `uint8_t red_state`
Target state of the light (on 1/off 0) represented as a boolean value.
- `uint8_t green_state`
Target state of the red color (0-255) represented as an 8-bit integer.
- `uint8_t blue_state`
Target state of the green color (0-255) represented as an 8-bit integer.
- `struct sensor_header header`
Target state of the blue color (0-255) represented as an 8-bit integer.
- `union sensor_data data`
Header of the packet containing length and type information.

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 27 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	

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 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 <stdint>
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 };
00026
00027 enum class PacketType : uint8_t {
00028     DATA = 0,
00029     HEARTBEAT = 1,
00030     DASHBOARD_POST = 2,
00031     DASHBOARD_GET = 3,
00032     DASHBOARD_RESPONSE = 4
00033 };
00034
00040 struct sensor_header {
00042     uint8_t length;
00044     PacketType ptype;
00045 } __attribute__((packed));
00046
00052 struct sensor_metadata {
```

```

00054     SensorType sensor_type;
00056     uint8_t sensor_id;
00057 } __attribute__((packed));
00058
00059 // Specific packet structures (ensure alignment/packing matches expected format)
00060
00069 struct sensor_heartbeat {
00070     struct sensor_metadata metadata;
00071 } __attribute__((packed));
00072
00081 struct sensor_packet_generic {
00082     struct sensor_metadata metadata;
00083     // /** @brief Whether the sensor did or did not trigger */
00084     // bool value;
00085 } __attribute__((packed));
00086
00094 struct sensor_packet_temperature {
00095     struct sensor_metadata metadata;
00097     float value;
00098 } __attribute__((packed));
00099
00107 struct sensor_packet_co2 {
00108     struct sensor_metadata metadata;
00110     uint16_t value;
00111 } __attribute__((packed));
00112
00120 struct sensor_packet_humidity {
00121     struct sensor_metadata metadata;
00123     float value;
00124 } __attribute__((packed));
00125
00133 struct sensor_packet_light {
00134     struct sensor_metadata metadata;
00136     uint8_t target_state;
00137 } __attribute__((packed));
00138
00147 struct sensor_packet_rgb_light {
00148     struct sensor_metadata metadata;
00150     uint8_t red_state;
00152     uint8_t green_state;
00154     uint8_t blue_state;
00155 } __attribute__((packed));
00156 // --- End Structures ---
00157
00222 struct sensor_packet {
00224     struct sensor_header header;
00225
00227     union sensor_data {
00228         struct sensor_heartbeat heartbeat;
00229         struct sensor_packet_generic generic;
00230         struct sensor_packet_temperature temperature;
00231         struct sensor_packet_co2 co2;
00232         struct sensor_packet_humidity humidity;
00233         struct sensor_packet_light light;
00234         struct sensor_packet_rgb_light rgb_light;
00235     } data;
00236 } __attribute__((packed));
00237
00238 #endif // PACKETS_H

```

8.5 include/slavemanager.h File Reference

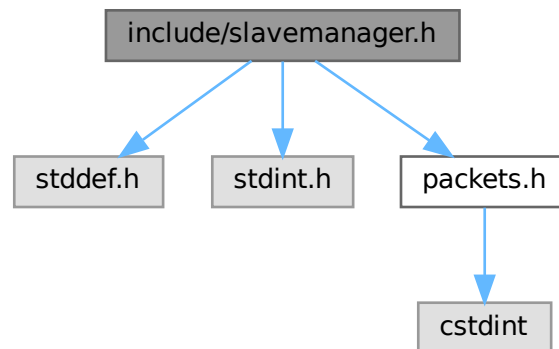
Header file for [slavemanager.cpp](#).

```

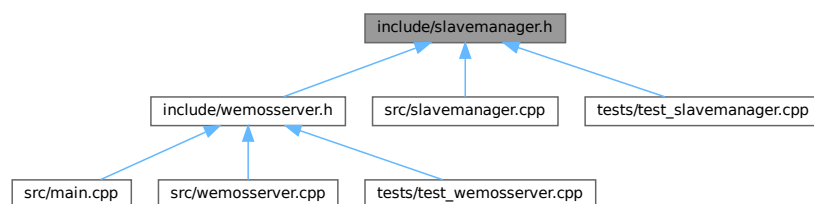
#include <stddef.h>
#include <stdint.h>
#include "packets.h"

```

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
00027 struct SlaveDevice {
00028     int fd;
00029     struct sensor_packet sensor_data;
00030 };
00031
00032 class SlaveManager {
00033     private:
00034         SlaveDevice slave_devices[MAX_SLAVE_ID + 1];
00035
00036     public:
00037         SlaveManager();
00038         ~SlaveManager();
00039
00040         SlaveManager(const SlaveManager &) = delete;
00041         SlaveManager &operator=(const SlaveManager &) = delete;
00042         SlaveManager(SlaveManager &&) = delete;
00043         SlaveManager &operator=(SlaveManager &&) = delete;
00044
00051         void registerSlave(uint8_t slave_id, int fd);
00052
00059         void unregisterSlave(uint8_t slave_id);
00060
00068         int sendToSlave(uint8_t slave_id, const void *data, size_t length);
00069
00075         int getSlaveFD(uint8_t slave_id) const;
00076
00082         SlaveDevice getSlaveDevice(uint8_t slave_id) const;
00083 };
00084
00085 #endif
```

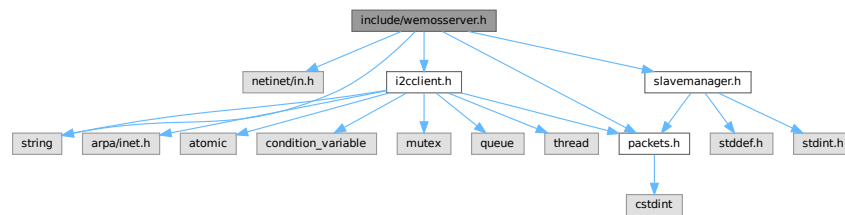
8.7 include/wemosserver.h File Reference

Header file for [wemosserver.cpp](#).

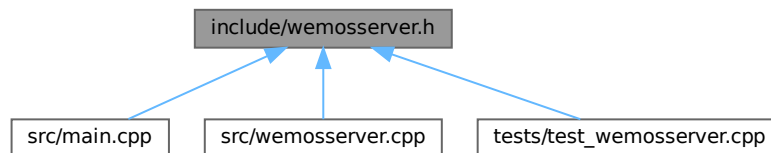
```
#include <netinet/in.h>
#include <string>
#include "i2cclient.h"
#include "packets.h"
```

```
#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 "i2ccclient.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         I2CCClient 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, uint8_t sensor_id);
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/i2ccclient.cpp File Reference

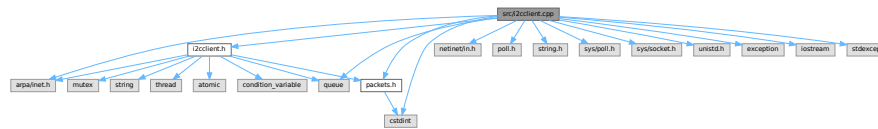
Implementation of [I2CCClient](#) class.

```

#include "i2ccclient.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 <stdint>
#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 I2CCClient::I2CCClient() : client_fd(-1), connected(false), running(false) {
00029     memset(&hub_address, 0, sizeof(hub_address));
00030 }
00031
00032 I2CCClient::~I2CCClient() {
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 I2CCClient::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 occurred, 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->p_type;
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                 queue_mutex.lock();
00124                 read_packets_queue.push(packet);
00125                 queue_mutex.unlock();
00126                 queue_condition.notify_one(); // maybe switch this with the line before if issues
00127                                             // occur - Erynn
00128
00129                 buffer_offset += sizeof(*head) + length;
00130             } while (buffer_offset + sizeof(struct sensor_header) <= amount_read);
00131         }
00132     }
00133
00134     std::terminate();
00135 }
00136
00137 void I2CClient::setup(const std::string &hub_ip, int hub_port) {
00138     if (inet_pton(AF_INET, hub_ip.c_str(), &hub_address.sin_addr) <= 0) {
00139         perror("inet_pton()");
00140         throw std::invalid_argument("Invalid IP address");
00141     }
00142
00143     if (hub_port <= 0 || hub_port > 65535) throw std::invalid_argument("Invalid port number");
00144
00145     hub_address.sin_family = AF_INET;
00146     hub_address.sin_port = htons(hub_port);
00147 }
00148
00149 bool I2CClient::openConnection() {
00150     if (client_fd >= 0) {
00151         close(client_fd);
00152         client_fd = -1;
00153     }
00154     connected = false;
00155
00156     std::string ip(inet_ntoa(hub_address.sin_addr));
00157     uint16_t port = ntohs(hub_address.sin_port);
00158
00159     std::cout << "Connecting to I2C hub at " << ip << ":" << port << std::endl;
00160
00161     client_fd = socket(AF_INET, SOCK_STREAM, 0);
00162     if (client_fd < 0) {
00163         std::cerr << "Socket creation failed" << std::endl;
00164         return false;
00165     }
00166
00167     if (connect(client_fd, (struct sockaddr *)&hub_address, sizeof(hub_address)) < 0) {
00168         int err = errno;
00169         std::cerr << "Connection failed: " << strerror(err) << std::endl;
00170         close(client_fd);
00171         client_fd = -1;
00172         return false;
00173     }
00174
00175     std::cout << "Connected to I2C hub at " << ip << ":" << port << std::endl;

```

```

00176     connected = true;
00177
00178     return true;
00179 }
00180
00181 void I2CClient::start() {
00182     if (!connected) {
00183         std::cerr
00184             « "Could not start communicating with I2C-bridge because not connected to I2C hub"
00185             « std::endl;
00186         throw std::runtime_error("Not connected to I2C-bridge");
00187     }
00188
00189     running = true;
00190     receive_thread = std::thread(&I2CClient::receiveLoop, this);
00191 }
00192
00193 void I2CClient::closeConnection() {
00194     if (!connected) {
00195         std::cerr « "Could not close the connection to I2C-bridge because not connected to I2C "
00196                 « "hub (either already closed, or never connected in the first place)"
00197                 « std::endl;
00198         return;
00199     }
00200
00201     running = false;
00202     receive_thread.join();
00203 }
00204
00205 void I2CClient::sendRawData(uint8_t *data, size_t length) {
00206     if (send(client_fd, data, length, 0) == -1) {
00207         perror("send() failed");
00208         throw std::runtime_error("Sending data to I2C-bridge failed");
00209     }
00210 }
00211
00212 struct sensor_packet I2CClient::retrievePacket(bool block) {
00213     queue_mutex.lock();
00214
00215     if (read_packets_queue.size() < 1 && !block) {
00216         // there are no packets available to retrieve,
00217         // and we're not blocking
00218         queue_mutex.unlock();
00219         throw std::runtime_error("No packet data available to retrieve from I2C-bridge");
00220     } else if (read_packets_queue.size() < 1) {
00221         // there are no packets available, but we block until there is
00222         std::unique_lock lk(queue_mutex);
00223         queue_condition.wait(lk);
00224     }
00225
00226     struct sensor_packet return_packet;
00227     memcpy(&return_packet, &read_packets_queue.front(), sizeof(return_packet));
00228     read_packets_queue.pop();
00229
00230     queue_mutex.unlock();
00231
00232     return return_packet;
00233 }

```

8.13 src/main.cpp File Reference

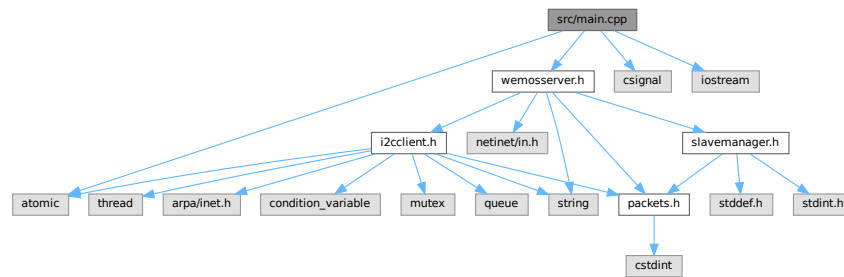
Main entrypoint for Wemos Bridge Server application.

```

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

```

Include dependency graph for main.cpp:



Macros

- `#define SERVER_PORT 5000`
- `#define I2C_HUB_IP "10.0.0.3"`
- `#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 "10.0.0.3"
```

Definition at line 13 of file [main.cpp](#).

8.13.2.2 I2C_HUB_PORT

```
#define I2C_HUB_PORT 5000
```

Definition at line 14 of file [main.cpp](#).

8.13.2.3 SERVER_PORT

```
#define SERVER_PORT 5000
```

Definition at line 12 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 25 of file [main.cpp](#).

8.13.3.3 signalHandler()

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

Definition at line 18 of file [main.cpp](#).

8.14 main.cpp

[Go to the documentation of this file.](#)

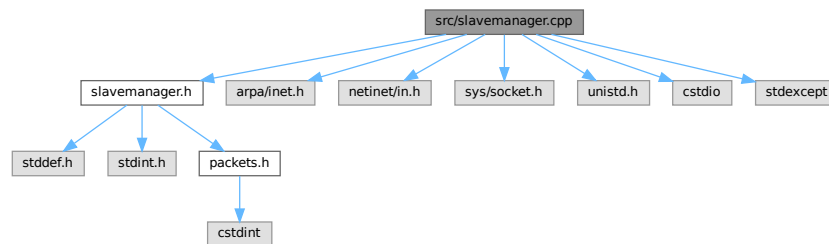
```
00001  
00006 #include <atomic>  
00007 #include <csignal>  
00008 #include <iostream>  
00009  
00010 #include "wemosserver.h"  
00011  
00012 #define SERVER_PORT 5000  
00013 #define I2C_HUB_IP "10.0.0.3"  
00014 #define I2C_HUB_PORT 5000  
00015  
00016 std::atomic<bool> global_shutdown_flag(false);  
00017  
00018 void signalHandler(int signum) {  
00019     std::cout << "Interrupt signal ( " << signum << " ) received.\n";  
00020     if (signum == SIGINT || signum == SIGTERM) {  
00021         global_shutdown_flag = true;  
00022     }  
00023 }  
00024  
00025 int main() {  
00026     setbuf(stdout, NULL);  
00027     std::cout << "Starting Wemos Bridge on port " << SERVER_PORT << std::endl;  
00028  
00029     // signal(SIGINT, signalHandler);  
00030     // signal(SIGTERM, signalHandler);  
00031  
00032     WemosServer server(SERVER_PORT, I2C_HUB_IP, I2C_HUB_PORT);  
00033     server.start();  
00034  
00035     return 0;  
00036 }
```

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 <stdexcept>
```

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 <stdexcept>
00016
00017 SlaveManager::SlaveManager() {
00018     for (int i = 0; i <= MAX_SLAVE_ID; ++i) {
00019         slave_devices[i].fd = -1;
00020     }
00021 }
00022
00023 SlaveManager::~SlaveManager() {
00024     for (int i = 0; i <= MAX_SLAVE_ID; ++i) {
00025         if (slave_devices[i].fd >= 0) {
```

```

00026         close(slave_devices[i].fd);
00027         slave_devices[i].fd = -1;
00028     }
00029 }
00030 }
00031
00032 void SlaveManager::registerSlave(uint8_t slave_id, int fd) {
00033     if (slave_id > MAX_SLAVE_ID || slave_id < 0) {
00034         printf("Invalid slave ID=%u\n", slave_id);
00035         throw std::invalid_argument("Invalid slave ID");
00036     }
00037
00038     printf("Registering new slave ID=%u\n", slave_id);
00039
00040     slave_devices[slave_id].fd = fd;
00041 }
00042
00043 void SlaveManager::unregisterSlave(uint8_t slave_id) {
00044     if (slave_id > MAX_SLAVE_ID || slave_id < 0) {
00045         printf("Invalid slave ID=%u\n", slave_id);
00046         throw std::invalid_argument("Invalid slave ID");
00047     }
00048
00049     printf("Unregistering slave ID=%u\n", slave_id);
00050     close(slave_devices[slave_id].fd);
00051     slave_devices[slave_id].fd = -1;
00052 }
00053
00054 int SlaveManager::sendToSlave(uint8_t slave_id, const void *data, size_t length) {
00055     if (slave_id > MAX_SLAVE_ID || slave_id < 0) {
00056         printf("Invalid slave ID=%u\n", slave_id);
00057         throw std::invalid_argument("Invalid slave ID");
00058     }
00059
00060     printf("Sending %zu bytes to slave ID=%u\n", length, slave_id);
00061     if (slave_devices[slave_id].fd < 0) {
00062         printf("Slave ID=%u not registered\n", slave_id);
00063         return -1;
00064     }
00065
00066     ssize_t bytes_sent = send(slave_devices[slave_id].fd, data, length, 0);
00067     if (bytes_sent < 0) {
00068         perror("send to slave failed");
00069         return -1;
00070     }
00071
00072     return 0;
00073 }
00074
00075 int SlaveManager::getSlaveFD(uint8_t slave_id) const {
00076     if (slave_id > MAX_SLAVE_ID || slave_id < 0) {
00077         printf("Invalid slave ID=%u\n", slave_id);
00078         throw std::invalid_argument("Invalid slave ID");
00079     }
00080
00081     return slave_devices[slave_id].fd;
00082 }
00083
00084 SlaveDevice SlaveManager::getSlaveDevice(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];
00091 }

```

8.17 src/wemosserver.cpp File Reference

Implementation of [WemosServer](#) class.

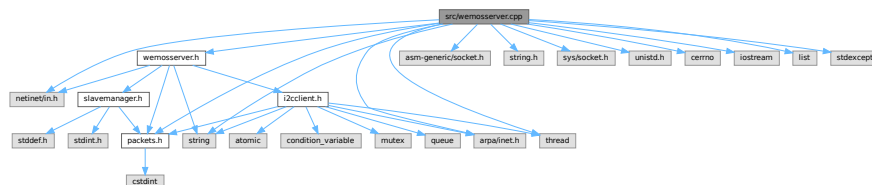
```

#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 <cerrno>
#include <iostream>
#include <list>
#include <stdexcept>
#include <string>
#include <thread>
#include "packets.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 100`
Maximum number of clients that can be connected to the server.

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 100
```

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

Definition at line 36 of file [wemosserver.cpp](#).

8.18 wemosserver.cpp

[Go to the documentation of this file.](#)

```
00001
00010 #include "wemosserver.h"
00011
00012 #include <arpa/inet.h>
00013 #include <asm-generic/socket.h>
00014 #include <netinet/in.h>
00015 #include <string.h>
00016 #include <sys/socket.h>
00017 #include <unistd.h>
00018
00019 #include <cerrno>
00020 #include <iostream>
00021 #include <list>
00022 #include <stdexcept>
00023 #include <string>
00024 #include <thread>
00025
00026 #include "packets.h"
00027
00031 #define BUFFER_SIZE 1024
00032
00036 #define MAX_CLIENTS 100
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
00061             if (offset + data_length + sizeof(struct sensor_header) > bytes_received) {
00062                 printf("Incomplete packet received, discarding\n");
00063                 break;
00064             }
00065
00066             switch (ptype) {
00067                 case PacketType::DATA:
00068                     printf("Packet length: %u, type: %u\n", data_length, s_type);
00069
00070                     processSensorData((const struct sensor_packet *)&buffer[offset]);
00071                     break;
00072
00073                 case PacketType::HEARTBEAT:
00074                     printf("Heartbeat packet: ID=%u, type=%u\n",
00075                           pkt_ptr->data.heartbeat.metadata.sensor_id,
00076                           pkt_ptr->data.heartbeat.metadata.sensor_type);
00077
00078                     // Register the slave device
00079                     slave_manager.registerSlave(pkt_ptr->data.heartbeat.metadata.sensor_id,
00080                                                 client_fd);
```

```

00081         break;
00082
00083     case PacketType::DASHBOARD_GET:
00084         printf("Dashboard requested data on sensor: ID=%u, type=%u",
00085             pkt_ptr->data.generic.metadata.sensor_id,
00086             pkt_ptr->data.generic.metadata.sensor_type);
00087
00088         sendToDashboard(client_fd, pkt_ptr->data.generic.metadata.sensor_id);
00089         break;
00090
00091     case PacketType::DASHBOARD_POST:
00092         // the dashboard is trying to update something
00093
00094         break;
00095
00096     default:
00097         // unknown packet type
00098         break;
00099 }
00100
00101 offset += data_length + sizeof(struct sensor_header);
00102 }
00103 }
00104
00105 if (bytes_received == 0) {
00106     printf("Connection closed by %s:%d\n", inet_ntoa(client_address.sin_addr),
00107         ntohs(client_address.sin_port));
00108 } else if (bytes_received < 0) {
00109     perror("recv failed");
00110 }
00111
00112 close(client_fd);
00113 }
00114
00115 void WemosServer::processSensorData(const struct sensor_packet *packet) {
00116     switch (packet->data.generic.metadata.sensor_type) {
00117     case SensorType::BUTTON: {
00118         printf("Processing button data: ID=%u\n", packet->data.generic.metadata.sensor_id);
00119
00120         // TODO: een tafel-knop ingedrukt
00121         /*
00122         if (btn->id == 0x69) {
00123             toggle_led(0x50);
00124             struct sensor_header header = {.length = sizeof(struct sensor_packet_light),
00125                 .type = DATA};
00126             struct sensor_packet_light led_control = {
00127                 .type = LIGHT, .id = 0x50, .target_state = get_led_state(0x50)};
00128
00129             uint8_t buffer[sizeof(struct sensor_header) + sizeof(struct sensor_packet_light)] =
00130                 {0};
00131             memcpy(buffer, &header, sizeof(header));
00132             memcpy(buffer + sizeof(header), &led_control, sizeof(led_control));
00133
00134             send_to_slave(0x50, &buffer, sizeof(buffer));
00135         }
00136
00137         if (btn->id == 0x70) {
00138             toggle_led(0x55);
00139             struct i2c_led_control led_control = {.led_number = 0x55,
00140                 .led_state = get_led_state(0x55)};
00141             send_to_rpi(&led_control, sizeof(led_control));
00142         }
00143         */
00144
00145         break;
00146     }
00147     case SensorType::TEMPERATURE: {
00148         printf("Processing temperature data: ID=%u, Value=%.2f\n",
00149             packet->data.temperature.metadata.sensor_id, packet->data.temperature.value);
00150
00151         // TODO: do temperature things
00152         break;
00153     }
00154     case SensorType::CO2: {
00155         printf("Processing CO2 data: ID=%u, Value=%u\n", packet->data.co2.metadata.sensor_id,
00156             packet->data.co2.value);
00157
00158         // TODO: do CO2 things
00159         break;
00160     }
00161     case SensorType::HUMIDITY: {
00162         printf("Processing humidity data: ID=%u, Value=%.2f\n",
00163             packet->data.humidity.metadata.sensor_id, packet->data.humidity.value);
00164
00165         // TODO: do humidity things
00166         break;
00167     }

```

```

00168         default:
00169             printf("No action defined for sensor type %u\n",
00170                 packet->data.generic.metadata.sensor_type);
00171             break;
00172     }
00173 }
00174
00175 void WemosServer::sendToDashboard(int dashboard_fd, uint8_t slave_id) {
00176     struct sensor_packet sensor_data;
00177
00178     sensor_data = slave_manager.getSlaveDevice(slave_id).sensor_data;
00179     sensor_data.header.ptype = PacketType::DASHBOARD_RESPONSE;
00180
00181     int length_to_send = sensor_data.header.length + sizeof(struct sensor_header);
00182
00183     send(dashboard_fd, &sensor_data, length_to_send, 0);
00184 }
00185 // private methods end here
00186
00187 WemosServer::WemosServer(int port, const std::string &hub_ip, int hub_port)
00188     : server_fd(-1), hub_ip(hub_ip), hub_port(hub_port), i2c_client() {
00189     if (port <= 0 || port > 65535) throw std::invalid_argument("Invalid listen port number");
00190
00191     if (INADDR_NONE == inet_addr(hub_ip.c_str()))
00192         throw std::invalid_argument("Invalid hub IP address passed");
00193     if (hub_port <= 0 || hub_port > 65535) throw std::invalid_argument("Invalid hub port passed");
00194
00195     memset(&listen_address, 0, sizeof(listen_address));
00196     listen_address.sin_family = AF_INET;
00197     listen_address.sin_addr = {INADDR_ANY};
00198     listen_address.sin_port = htons(port);
00199 }
00200
00201 WemosServer::~WemosServer() {
00202     tearDown();
00203     // other shit
00204 }
00205
00206 void WemosServer::socketSetup() {
00207     if ((server_fd = socket(AF_INET, SOCK_STREAM | SOCK_NONBLOCK, 0)) < 0) {
00208         perror("socket() failed");
00209         throw std::runtime_error("socket() failed");
00210         exit(EXIT_FAILURE);
00211     }
00212
00213     const int enable_opt = 1;
00214     if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &enable_opt,
00215         sizeof(enable_opt)) < 0) {
00216         perror("setsockopt() failed");
00217         throw std::runtime_error("setsockopt() failed");
00218         exit(EXIT_FAILURE);
00219     }
00220
00221     if (bind(server_fd, (struct sockaddr *)&listen_address, sizeof(listen_address)) < 0) {
00222         perror("bind() failed");
00223         throw std::runtime_error("bind() failed");
00224         exit(EXIT_FAILURE);
00225     }
00226
00227     if (listen(server_fd, MAX_CLIENTS) < 0) {
00228         perror("listen() failed");
00229         throw std::runtime_error("listen() failed");
00230         exit(EXIT_FAILURE);
00231     }
00232
00233     std::cout << "Listening on port " << ntohs(listen_address.sin_port) << " (max " << MAX_CLIENTS
00234         << " clients)" << std::endl;
00235 }
00236
00237 void WemosServer::setupI2cClient() { i2c_client.setup(hub_ip, hub_port); }
00238
00239 void WemosServer::start() {
00240     socketSetup();
00241
00242     setupI2cClient();
00243     i2c_client.openConnection();
00244     i2c_client.start();
00245
00246     while (true) {
00247         struct sensor_packet pkt;
00248         try {
00249             pkt = i2c_client.retrievePacket();
00250             // std::cout << "packet received from the I2C hub!" << std::endl;
00251
00252             // now do things with the I2C packet if necessary
00253
00254         } catch (std::runtime_error &exc) {

```

```

00255         /* this means there is no new I2C packet available */
00256         // std::cerr << exc.what() << std::endl;
00257     }
00258
00259     struct sockaddr_in client_address;
00260     socklen_t client_addr_len = sizeof(client_address);
00261     int client_fd = accept(server_fd, (struct sockaddr *)&client_address, &client_addr_len);
00262
00263     if (-1 == client_fd) {
00264         // no one tried to connect
00265         continue;
00266     }
00267
00268     std::cout << "Connection accepted from " << inet_ntoa(client_address.sin_addr) << ":"
00269             << ntohs(client_address.sin_port) << std::endl;
00270
00271     handleClient(client_fd, client_address);
00272 }
00273 }
00274
00275 void WemosServer::tearDown() {
00276     close(server_fd);
00277     i2c_client.closeConnection();
00278 }

```

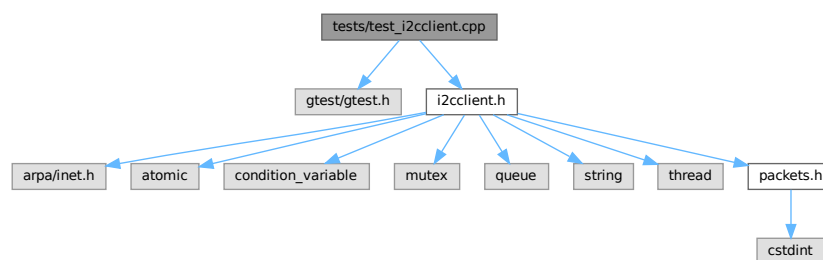
8.19 tests/test_i2cclient.cpp File Reference

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(I2CCClientTests, setup_ValidPort) {
00019     I2CCClient 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(I2CCClientTests, setup_InvalidPort_Negative) {
00033     I2CCClient server;
00034     EXPECT_THROW(server.setup("10.0.0.1", -1), std::invalid_argument);
00035 }
00036
00044 TEST(I2CCClientTests, setup_InvalidPort_Zero) {
00045     I2CCClient server;
00046     EXPECT_THROW(server.setup("10.0.0.1", 0), std::invalid_argument);
00047 }
00048
00057 TEST(I2CCClientTests, setup_InvalidPort_High) {
00058     I2CCClient 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 }

```

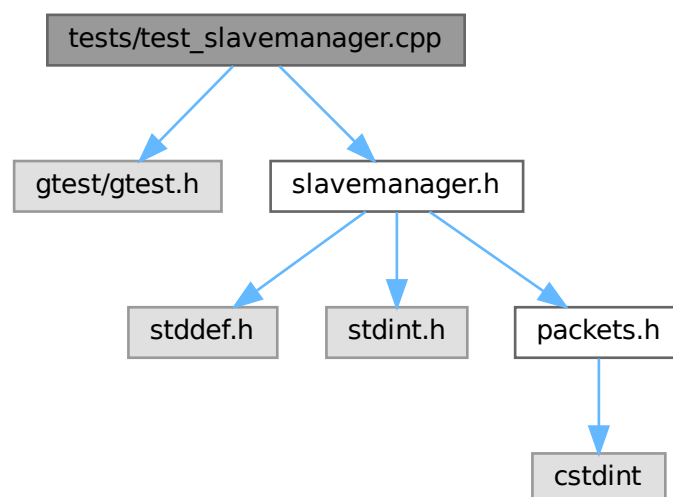
8.21 tests/test_slavemanager.cpp File Reference

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 19 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     EXPECT_EQ(manager.getSlaveDevice(1).fd, fd);
00017 }
00018
00019 TEST(SlaveManagerTests, UnregisterSlave) {
00020     SlaveManager manager;
00021     int fd = 5;
00022
00023     EXPECT_NO_THROW(manager.registerSlave(1, fd));
00024     EXPECT_EQ(manager.getSlaveFD(1), fd);
00025
00026     EXPECT_NO_THROW(manager.unregisterSlave(1));
00027     EXPECT_EQ(manager.getSlaveFD(1), -1);
00028     EXPECT_EQ(manager.getSlaveDevice(1).fd, -1);
00029 }
```

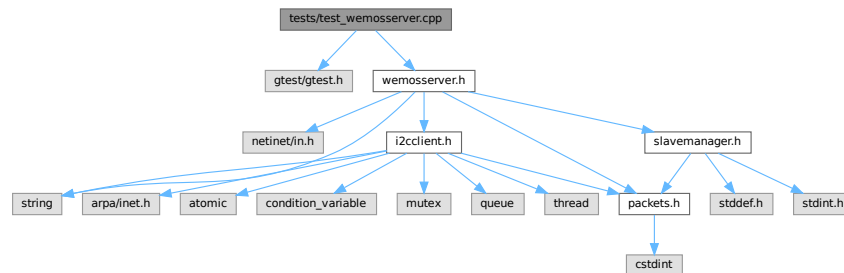
8.23 tests/test_wemosserver.cpp File Reference

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 too 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.Constructor_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

- __attribute__
 - packets.h, 58
- ~I2CClient
 - I2CClient, 20
- ~SlaveManager
 - SlaveManager, 44
- ~WemosServer
 - WemosServer, 49
- blue_state
 - packets.h, 58
 - sensor_packet_rgb_light, 40
- BUFFER_SIZE
 - i2cclient.cpp, 66
 - wemosserver.cpp, 74
- BUTTON
 - packets.h, 58
- client_fd
 - I2CClient, 23
- closeConnection
 - I2CClient, 20
- CO2
 - packets.h, 58
- co2
 - sensor_data, 25
 - sensor_packet::sensor_data, 27
- connected
 - I2CClient, 23
- DASHBOARD_GET
 - packets.h, 57
- DASHBOARD_POST
 - packets.h, 57
- DASHBOARD_RESPONSE
 - packets.h, 57
- DATA
 - packets.h, 57
- data
 - I2CClient::DataReceiveReturn, 18
 - packets.h, 58
 - sensor_packet, 33
- fd
 - SlaveDevice, 43
- generic
 - sensor_data, 25
 - sensor_packet::sensor_data, 27
- getSlaveDevice
 - SlaveManager, 45
- getSlaveFD
 - SlaveManager, 45
- global_shutdown_flag
 - main.cpp, 71
- green_state
 - packets.h, 58
 - sensor_packet_rgb_light, 40
- handleClient
 - WemosServer, 50
- header
 - packets.h, 58
 - sensor_packet, 33
- HEARTBEAT
 - packets.h, 57
- heartbeat
 - sensor_data, 25
 - sensor_packet::sensor_data, 27
- hub_address
 - I2CClient, 23
- hub_ip
 - WemosServer, 51
- hub_port
 - WemosServer, 51
- HUMIDITY
 - packets.h, 58
- humidity
 - sensor_data, 25
 - sensor_packet::sensor_data, 27
- i2c_client
 - WemosServer, 52
- I2C_HUB_IP
 - main.cpp, 70
- I2C_HUB_PORT
 - main.cpp, 70
- I2CClient, 18
 - ~I2CClient, 20
 - client_fd, 23
 - closeConnection, 20
 - connected, 23
 - hub_address, 23
 - I2CClient, 19, 20
 - openConnection, 20
 - operator=, 20, 21
 - queue_condition, 23
 - queue_mutex, 23
 - read_packets_queue, 23
 - receive_mutex, 24
 - receive_thread, 24

- receiveLoop, 21
- retrievePacket, 21
- running, 24
- sendRawData, 22
- setup, 22
- start, 23
- i2ccclient.cpp
 - BUFFER_SIZE, 66
 - THREAD_RELINQUISH, 66
- I2CCClient::DataReceiveReturn, 17
 - data, 18
 - length, 18
- I2CCClientTests, 12
 - TEST, 13
- include/i2ccclient.h, 53, 54
- include/packets.h, 55, 60
- include/slavemanager.h, 61, 63
- include/wemosserver.h, 63, 65
- length
 - I2CCClient::DataReceiveReturn, 18
 - packets.h, 59
 - sensor_header, 29
- LIGHT
 - packets.h, 58
- light
 - sensor_data, 25
 - sensor_packet::sensor_data, 27
- listen_address
 - WemosServer, 52
- main
 - main.cpp, 71
- main.cpp
 - global_shutdown_flag, 71
 - I2C_HUB_IP, 70
 - I2C_HUB_PORT, 70
 - main, 71
 - SERVER_PORT, 70
 - signalHandler, 71
- MAX_CLIENTS
 - wemosserver.cpp, 74
- MAX_SLAVE_ID
 - slavemanager.h, 63
- metadata
 - packets.h, 59
 - sensor_heartbeat, 30
 - sensor_packet_co2, 34
 - sensor_packet_generic, 36
 - sensor_packet_humidity, 37
 - sensor_packet_light, 38
 - sensor_packet_rgb_light, 40
 - sensor_packet_temperature, 42
- modules.dox, 65
- MOTION
 - packets.h, 58
- NOOP
 - packets.h, 58
- openConnection
 - I2CCClient, 20
- operator=
 - I2CCClient, 20, 21
 - SlaveManager, 45
 - WemosServer, 50
- Packets, 14
- packets.h
 - __attribute__, 58
 - blue_state, 58
 - BUTTON, 58
 - CO2, 58
 - DASHBOARD_GET, 57
 - DASHBOARD_POST, 57
 - DASHBOARD_RESPONSE, 57
 - DATA, 57
 - data, 58
 - green_state, 58
 - header, 58
 - HEARTBEAT, 57
 - HUMIDITY, 58
 - length, 59
 - LIGHT, 58
 - metadata, 59
 - MOTION, 58
 - NOOP, 58
 - PacketType, 57
 - PRESSURE, 58
 - ptype, 59
 - red_state, 59
 - RGB_LIGHT, 58
 - sensor_id, 59
 - sensor_type, 59
 - SensorType, 57
 - target_state, 60
 - TEMPERATURE, 58
 - value, 60
- PacketType
 - packets.h, 57
- PRESSURE
 - packets.h, 58
- processSensorData
 - WemosServer, 50
- ptype
 - packets.h, 59
 - sensor_header, 29
- queue_condition
 - I2CCClient, 23
- queue_mutex
 - I2CCClient, 23
- read_packets_queue
 - I2CCClient, 23
- README.md, 65
- receive_mutex
 - I2CCClient, 24
- receive_thread

- I2CCClient, [24](#)
- receiveLoop
 - I2CCClient, [21](#)
- red_state
 - packets.h, [59](#)
 - sensor_packet_rgb_light, [40](#)
- registerSlave
 - SlaveManager, [46](#)
- retrievePacket
 - I2CCClient, [21](#)
- RGB_LIGHT
 - packets.h, [58](#)
- rgb_light
 - sensor_data, [25](#)
 - sensor_packet::sensor_data, [27](#)
- running
 - I2CCClient, [24](#)
- sendRawData
 - I2CCClient, [22](#)
- sendToDashboard
 - WemosServer, [50](#)
- sendToSlave
 - SlaveManager, [46](#)
- sensor_data, [24](#)
 - co2, [25](#)
 - generic, [25](#)
 - heartbeat, [25](#)
 - humidity, [25](#)
 - light, [25](#)
 - rgb_light, [25](#)
 - SlaveDevice, [43](#)
 - temperature, [26](#)
- sensor_header, [28](#)
 - length, [29](#)
 - ptype, [29](#)
- sensor_heartbeat, [29](#)
 - metadata, [30](#)
- sensor_id
 - packets.h, [59](#)
 - sensor_metadata, [31](#)
- sensor_metadata, [30](#)
 - sensor_id, [31](#)
 - sensor_type, [31](#)
- sensor_packet, [31](#)
 - data, [33](#)
 - header, [33](#)
- sensor_packet::sensor_data, [26](#)
 - co2, [27](#)
 - generic, [27](#)
 - heartbeat, [27](#)
 - humidity, [27](#)
 - light, [27](#)
 - rgb_light, [27](#)
 - temperature, [27](#)
- sensor_packet_co2, [33](#)
 - metadata, [34](#)
 - value, [34](#)
- sensor_packet_generic, [35](#)
 - metadata, [36](#)
- sensor_packet_humidity, [36](#)
 - metadata, [37](#)
 - value, [37](#)
- sensor_packet_light, [37](#)
 - metadata, [38](#)
 - target_state, [38](#)
- sensor_packet_rgb_light, [39](#)
 - blue_state, [40](#)
 - green_state, [40](#)
 - metadata, [40](#)
 - red_state, [40](#)
- sensor_packet_temperature, [41](#)
 - metadata, [42](#)
 - value, [42](#)
- sensor_type
 - packets.h, [59](#)
 - sensor_metadata, [31](#)
- SensorType
 - packets.h, [57](#)
- server_fd
 - WemosServer, [52](#)
- SERVER_PORT
 - main.cpp, [70](#)
- setup
 - I2CCClient, [22](#)
- setupI2cClient
 - WemosServer, [50](#)
- signalHandler
 - main.cpp, [71](#)
- slave_devices
 - SlaveManager, [47](#)
- slave_manager
 - WemosServer, [52](#)
- SlaveDevice, [42](#)
 - fd, [43](#)
 - sensor_data, [43](#)
- SlaveManager, [43](#)
 - ~SlaveManager, [44](#)
 - getSlaveDevice, [45](#)
 - getSlaveFD, [45](#)
 - operator=, [45](#)
 - registerSlave, [46](#)
 - sendToSlave, [46](#)
 - slave_devices, [47](#)
 - SlaveManager, [44](#)
 - unregisterSlave, [46](#)
- slavemanager.h
 - MAX_SLAVE_ID, [63](#)
- SlaveManagerTests, [14](#)
- socketSetup
 - WemosServer, [51](#)
- src/i2ccclient.cpp, [65](#), [67](#)
- src/main.cpp, [69](#), [71](#)
- src/slavemanager.cpp, [72](#)
- src/wemosserver.cpp, [73](#), [75](#)
- start
 - I2CCClient, [23](#)

- WemosServer, [51](#)
- target_state
 - packets.h, [60](#)
 - sensor_packet_light, [38](#)
- tearDown
 - WemosServer, [51](#)
- TEMPERATURE
 - packets.h, [58](#)
- temperature
 - sensor_data, [26](#)
 - sensor_packet::sensor_data, [27](#)
- TEST
 - I2CClientTests, [13](#)
 - test_slavemanager.cpp, [80](#)
 - test_wemosserver.cpp, [82–84](#)
- Test List, [3](#)
- test_slavemanager.cpp
 - TEST, [80](#)
- test_wemosserver.cpp
 - TEST, [82–84](#)
- Tests, [11](#)
- tests/test_i2cclient.cpp, [78, 79](#)
- tests/test_slavemanager.cpp, [79, 80](#)
- tests/test_wemosserver.cpp, [81, 85](#)
- THREAD_RELINQUISH
 - i2cclient.cpp, [66](#)
- unregisterSlave
 - SlaveManager, [46](#)
- value
 - packets.h, [60](#)
 - sensor_packet_co2, [34](#)
 - sensor_packet_humidity, [37](#)
 - sensor_packet_temperature, [42](#)
- Wemos Bridge Server, [1](#)
- WemosServer, [47](#)
 - ~WemosServer, [49](#)
 - handleClient, [50](#)
 - hub_ip, [51](#)
 - hub_port, [51](#)
 - i2c_client, [52](#)
 - listen_address, [52](#)
 - operator=, [50](#)
 - processSensorData, [50](#)
 - sendToDashboard, [50](#)
 - server_fd, [52](#)
 - setupI2cClient, [50](#)
 - slave_manager, [52](#)
 - socketSetup, [51](#)
 - start, [51](#)
 - tearDown, [51](#)
 - WemosServer, [49, 50](#)
- wemosserver.cpp
 - BUFFER_SIZE, [74](#)
 - MAX_CLIENTS, [74](#)
- WemosServerTests, [12](#)