Bus Bridge Server

commit-0c1a9b4

Generated by Doxygen 1.9.8

Chapter 1

Bus Bridge Server

Chapter 2

Test List

File test_math.cpp

MathTest.Add

- Verifies that the `add` function correctly computes the sum of two integers.
- Example: `add(2, 3)` should return `5`.

MathTest.Subtract

- Verifies that the `subtract` function correctly computes the difference between two integers.
- Examples:
  - `subtract(10, 3)` should return `7`.
  - `subtract(9, 3)` should return `6`.

  MathTest.SubtractNegative
- Verifies that the `subtract` function handles subtraction with negative integers correctly.
- Example: `subtract(10, -3)` should return `13`.

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

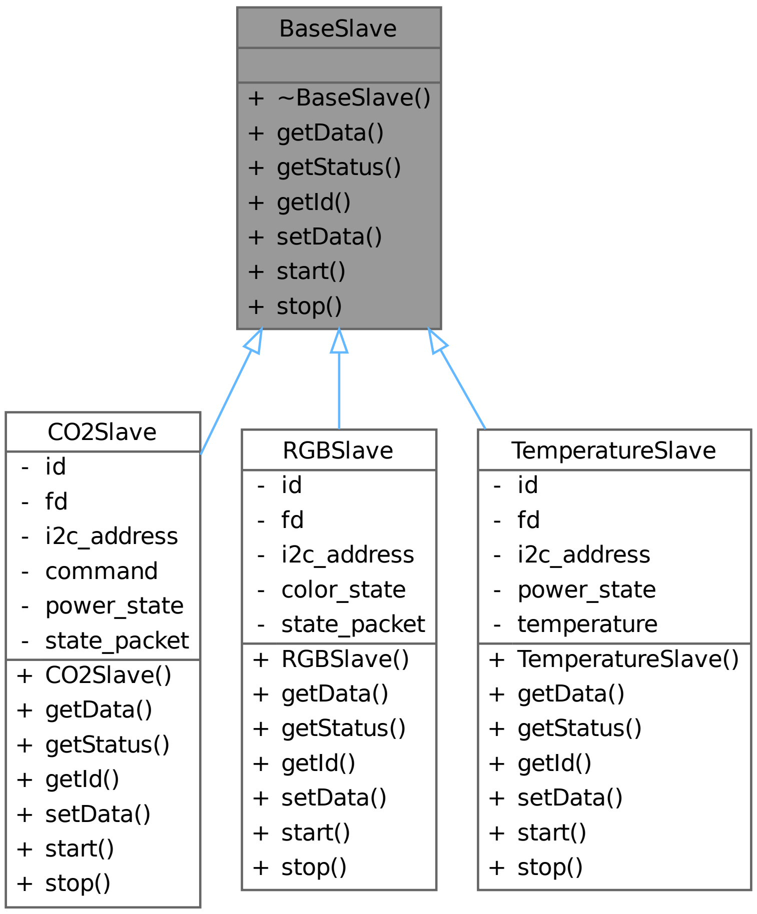
Chapter 6

Class Documentation

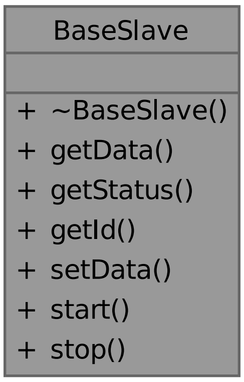
6.1 BaseSlave Class Reference

```
#include <BaseSlave.h>
```

Inheritance diagram for BaseSlave:



Collaboration diagram for BaseSlave:



Public Member Functions

- virtual ∼BaseSlave ()=default
- virtual const void ∗ getData ()=0
- virtual bool getStatus ()=0
- virtual int getId ()=0
- virtual void setData (void ∗data)=0
- virtual void start (int i2c_fd)=0
- virtual void stop ()=0

6.1.1 Detailed Description

Definition at line 6 of file BaseSlave.h.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 ∼BaseSlave()

```
virtual BaseSlave::∼BaseSlave ( )  [virtual], [default]
```

6.1.3 Member Function Documentation

6.1.3.1 getData()

```
virtual const void * BaseSlave::getData ( )  [pure virtual]
```

Implemented in RGBSlave, TemperatureSlave, and CO2Slave.

6.1.3.2 getId()

```
virtual int BaseSlave::getId ( )  [pure virtual]
```

Implemented in RGBSlave, TemperatureSlave, and CO2Slave.

6.1.3.3 getStatus()

```
virtual bool BaseSlave::getStatus ( )  [pure virtual]
```

Implemented in RGBSlave, TemperatureSlave, and CO2Slave.

6.1.3.4 setData()

```
virtual void BaseSlave::setData (
             void * data )  [pure virtual]
```

Implemented in RGBSlave, TemperatureSlave, and CO2Slave.

6.1.3.5 start()

```
virtual void BaseSlave::start (
             int i2c_fd )  [pure virtual]
```

Implemented in RGBSlave, TemperatureSlave, and CO2Slave.

6.1.3.6 stop()

```
virtual void BaseSlave::stop ( )  [pure virtual]
```

Implemented in RGBSlave, TemperatureSlave, and CO2Slave.

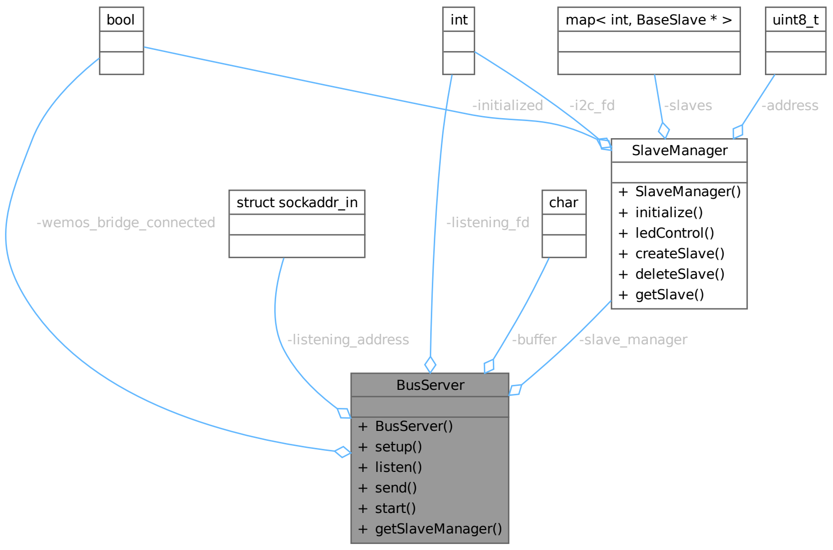
The documentation for this class was generated from the following file:

- include/BaseSlave.h

6.2 BusServer Class Reference

```
#include <BusServer.h>
```

Collaboration diagram for BusServer:



Public Member Functions

- BusServer ()
- void setup (std::string ip, int port)

  Setup for the IP socket and the I2C slaves connection.
- void listen ()

  Open the underlying socket for incoming connections.
- void send (struct sensor_packet *pkt, int fd)

  Send a sensor packet to a connected network client.
- void start ()

  Start the main loop of the BusServer.
- SlaveManager & getSlaveManager ()

  balls

Private Attributes

- int listening_fd
- struct sockaddr_in listening_address
- bool wemos_bridge_connected = false
- char buffer [BUFFER_SIZE]
- SlaveManager slave_manager

6.2.1 Detailed Description

Definition at line 12 of file BusServer.h.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 BusServer()

```
BusServer::BusServer ( )  [inline]
```

Definition at line 14 of file BusServer.h.

6.2.3 Member Function Documentation

6.2.3.1 getSlaveManager()

```
SlaveManager & BusServer::getSlaveManager ( )
```

balls

Definition at line 185 of file BusServer.cpp.

6.2.3.2 listen()

```
void BusServer::listen ( )
```

Open the underlying socket for incoming connections.

Exceptions

std::runtime_error	if the listening fails

Definition at line 60 of file BusServer.cpp.

6.2.3.3 send()

```
void BusServer::send (
             struct sensor_packet * pkt,
             int fd )
```

Send a sensor packet to a connected network client.

Depending on the data set in the packet header, send a certain amount of data to the client.

Parameters

pkt	A pointer to the sensor_packet struct to send over
fd	The file descriptor to send the packet over

Exceptions

std::runtime_error	if the sending fails for any reason

Definition at line 67 of file BusServer.cpp.

6.2.3.4 setup()

```
void BusServer::setup (
            std::string ip,
            int port )
```

Setup for the IP socket and the I2C slaves connection.

This method will set up a socket for listening on the network and will also tell the underlying SlaveManager object to initialize its I2C bus

Parameters

ip	The IP address to listen on within the network
port	The TCP port to listen on

Exceptions

std::invalid_argument	if the passed IP address or port number are invalid
std::runtime_error	if the socket creation fails

Definition at line 20 of file BusServer.cpp.

6.2.3.5 start()

```
void BusServer::start ( )
```

Start the main loop of the BusServer.

This will first initialize the underlying I2C connections to the directly-connected slave devices, and then start accepting and processing network clients

Definition at line 77 of file BusServer.cpp.

6.2.4 Member Data Documentation

6.2.4.1 buffer

```
char BusServer::buffer[BUFFER_SIZE]  [private]
```

Definition at line 60 of file BusServer.h.

6.2.4.2 listening_address

```
struct sockaddr_in BusServer::listening_address  [private]
```

Definition at line 58 of file BusServer.h.

6.2.4.3 listening_fd

```
int BusServer::listening_fd  [private]
```

Definition at line 56 of file BusServer.h.

6.2.4.4 slave_manager

```
SlaveManager BusServer::slave_manager  [private]
```

Definition at line 62 of file BusServer.h.

6.2.4.5 wemos_bridge_connected

```
bool BusServer::wemos_bridge_connected = false  [private]
```

Definition at line 59 of file BusServer.h.

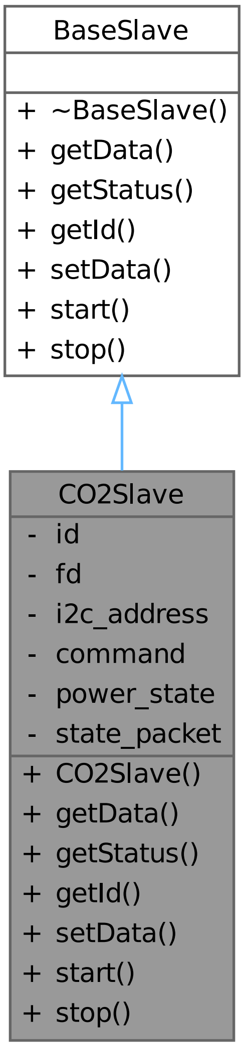
The documentation for this class was generated from the following files:

- include/BusServer.h
- src/BusServer.cpp

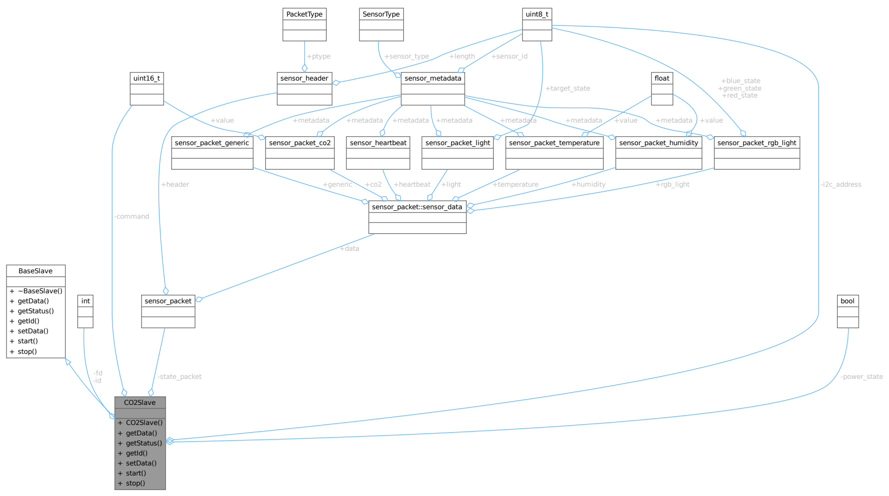
6.3 CO2Slave Class Reference

```
#include <CO2Slave.h>
```

Inheritance diagram for CO2Slave:



Collaboration diagram for CO2Slave:



Public Member Functions

- CO2Slave (uint8_t id, uint8_t i2c_address)
- const void ∗ getData () override
- bool getStatus () override
- int getId () override
- void setData (void ∗data) override
- void start (int i2c_fd) override
- void stop () override

Public Member Functions inherited from BaseSlave

- virtual ∼BaseSlave ()=default

Private Attributes

- int id
- int fd
- uint8_t i2c_address
- uint16_t command
- bool power_state = true
- struct sensor_packet state_packet

6.3.1 Detailed Description

Definition at line 7 of file CO2Slave.h.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 CO2Slave()

```
CO2Slave::CO2Slave (
             uint8_t id,
             uint8_t i2c_address )
```

Definition at line 5 of file CO2Slave.cpp.

6.3.3 Member Function Documentation

6.3.3.1 getData()

```
const void * CO2Slave::getData ( )  [override], [virtual]
```

Implements BaseSlave.

Definition at line 18 of file CO2Slave.cpp.

6.3.3.2 getId()

```
int CO2Slave::getId ( )  [override], [virtual]
```

Implements BaseSlave.

Definition at line 33 of file CO2Slave.cpp.

6.3.3.3 getStatus()

```
bool CO2Slave::getStatus ( )  [override], [virtual]
```

Implements BaseSlave.

Definition at line 31 of file CO2Slave.cpp.

6.3.3.4 setData()

```
void CO2Slave::setData (
             void * data )  [override], [virtual]
```

Implements BaseSlave.

Definition at line 35 of file CO2Slave.cpp.

6.3.3.5 start()

```
void CO2Slave::start (
             int i2c_fd )  [override], [virtual]
```

Implements BaseSlave.

Definition at line 37 of file CO2Slave.cpp.

6.3.3.6 stop()

```
void CO2Slave::stop ( )  [override], [virtual]
```

Implements BaseSlave.

Definition at line 39 of file CO2Slave.cpp.

6.3.4 Member Data Documentation

6.3.4.1 command

```
uint16_t CO2Slave::command  [private]
```

Definition at line 22 of file CO2Slave.h.

6.3.4.2 fd

```
int CO2Slave::fd  [private]
```

Definition at line 20 of file CO2Slave.h.

6.3.4.3 i2c_address

```
uint8_t CO2Slave::i2c_address  [private]
```

Definition at line 21 of file CO2Slave.h.

6.3.4.4 id

```
int CO2Slave::id  [private]
```

Definition at line 19 of file CO2Slave.h.

6.3.4.5 power_state

```
bool CO2Slave::power_state = true  [private]
```

Definition at line 24 of file CO2Slave.h.

6.3.4.6 state_packet

```
struct sensor_packet CO2Slave::state_packet  [private]
```

Definition at line 25 of file CO2Slave.h.

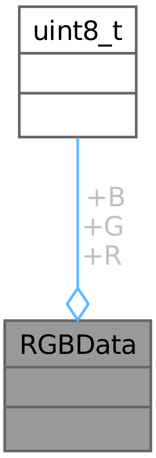
The documentation for this class was generated from the following files:

- include/CO2Slave.h
- src/CO2Slave.cpp

6.4 RGBData Struct Reference

```
#include <RGBSlave.h>
```

Collaboration diagram for RGBData:



Public Attributes

- uint8_t R
- uint8_t G
- uint8_t B

6.4.1 Detailed Description

Definition at line 7 of file RGBSlave.h.

6.4.2 Member Data Documentation

6.4.2.1 B

```
uint8_t RGBData::B
```

Definition at line 8 of file RGBSlave.h.

6.4.2.2 G

```
uint8_t RGBData::G
```

Definition at line 8 of file RGBSlave.h.

6.4.2.3 R

```
uint8_t RGBData::R
```

Definition at line 8 of file RGBSlave.h.

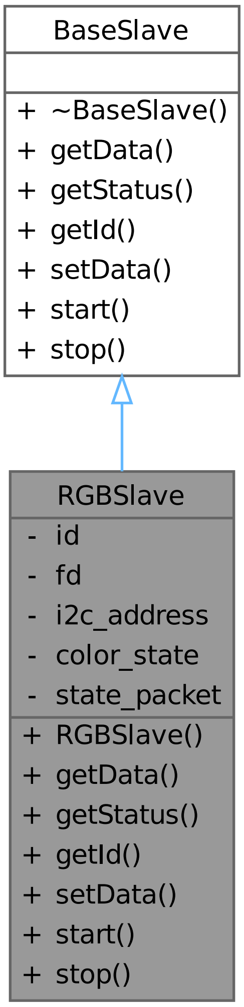
The documentation for this struct was generated from the following file:

- include/RGBSlave.h

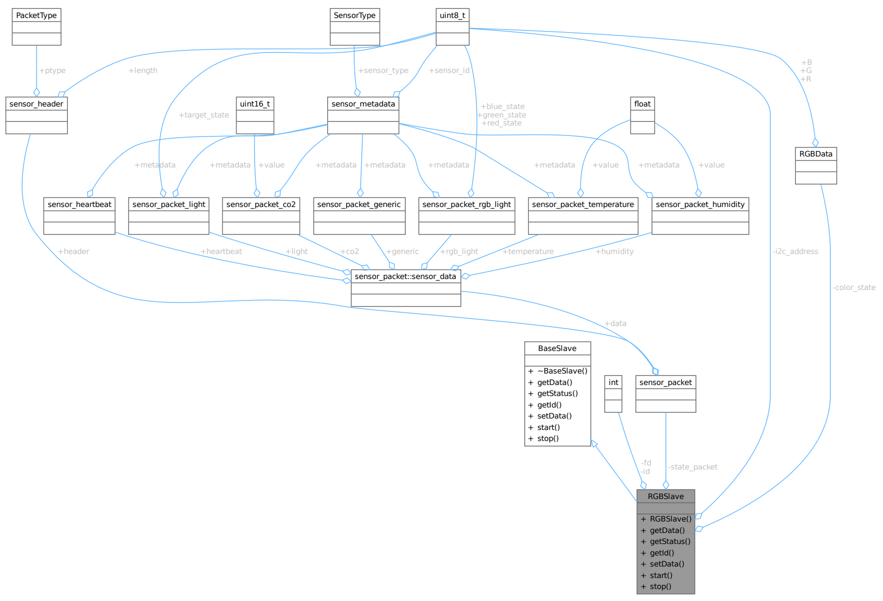
6.5 RGBSlave Class Reference

```
#include <RGBSlave.h>
```

Inheritance diagram for RGBSlave:



Collaboration diagram for RGBSlave:



Public Member Functions

- RGBSlave (uint8_t id, uint8_t i2c_address)
- const void ∗ getData ()
- bool getStatus ()
- int getId ()
- void setData (void ∗data)
- void start (int i2c_fd)
- void stop ()

Public Member Functions inherited from BaseSlave

- virtual ∼BaseSlave ()=default

Private Attributes

- int id
- int fd
- uint8_t i2c_address
- RGBData color_state
- struct sensor_packet state_packet

6.5.1 Detailed Description

Definition at line 11 of file RGBSlave.h.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 RGBSlave()

```
RGBSlave::RGBSlave (
             uint8_t id,
             uint8_t i2c_address )
```

Definition at line 7 of file RGBSlave.cpp.

6.5.3 Member Function Documentation

6.5.3.1 getData()

```
const void * RGBSlave::getData ( )  [virtual]
```

Implements BaseSlave.

Definition at line 15 of file RGBSlave.cpp.

6.5.3.2 getId()

```
int RGBSlave::getId ( )  [virtual]
```

Implements BaseSlave.

Definition at line 42 of file RGBSlave.cpp.

6.5.3.3 getStatus()

```
bool RGBSlave::getStatus ( )  [virtual]
```

Implements BaseSlave.

Definition at line 33 of file RGBSlave.cpp.

6.5.3.4 setData()

```
void RGBSlave::setData (
             void * data )  [virtual]
```

Implements BaseSlave.

Definition at line 44 of file RGBSlave.cpp.

6.5.3.5 start()

```
void RGBSlave::start (
             int i2c_fd )  [virtual]
```

Implements BaseSlave.

Definition at line 60 of file RGBSlave.cpp.

6.5.3.6 stop()

```
void RGBSlave::stop ( )  [virtual]
```

Implements BaseSlave.

Definition at line 62 of file RGBSlave.cpp.

6.5.4 Member Data Documentation

6.5.4.1 color_state

```
RGBData RGBSlave::color_state  [private]
```

Definition at line 26 of file RGBSlave.h.

6.5.4.2 fd

```
int RGBSlave::fd  [private]
```

Definition at line 23 of file RGBSlave.h.

6.5.4.3 i2c_address

```
uint8_t RGBSlave::i2c_address  [private]
```

Definition at line 24 of file RGBSlave.h.

6.5.4.4 id

```
int RGBSlave::id  [private]
```

Definition at line 22 of file RGBSlave.h.

6.5.4.5 state_packet

```
struct sensor_packet RGBSlave::state_packet  [private]
```

Definition at line 28 of file RGBSlave.h.

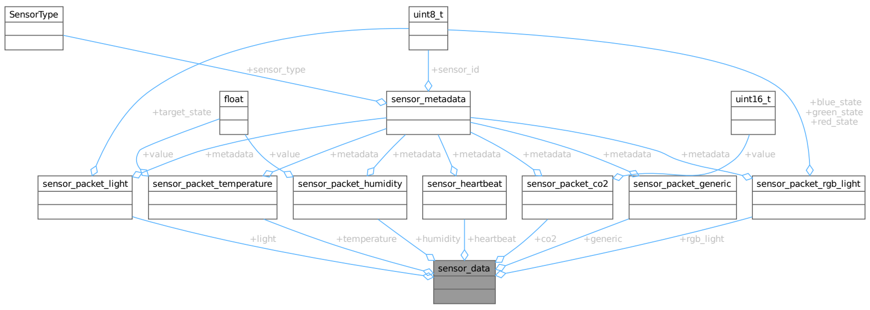
The documentation for this class was generated from the following files:

- include/RGBSlave.h
- src/RGBSlave.cpp

6.6 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

6.6.1 Detailed Description

Definition at line 4 of file packets.h.

6.6.2 Member Data Documentation

6.6.2.1 co2

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

Definition at line 8 of file packets.h.

6.6.2.2 generic

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

Definition at line 6 of file packets.h.

6.6.2.3 heartbeat

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

Definition at line 5 of file packets.h.

6.6.2.4 humidity

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

Definition at line 9 of file packets.h.

6.6.2.5 light

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

Definition at line 10 of file packets.h.

6.6.2.6 rgb_light

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

Definition at line 11 of file packets.h.

6.6.2.7 temperature

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

Definition at line 7 of file packets.h.

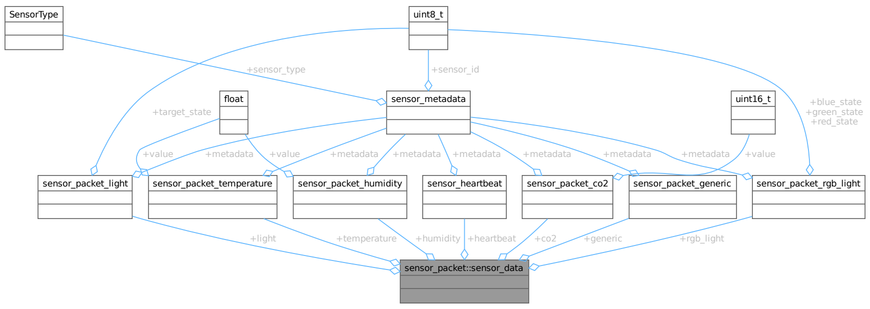
The documentation for this union was generated from the following file:

- include/packets.h

6.7 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

6.7.1 Detailed Description

Definition at line 227 of file packets.h.

6.7.2 Member Data Documentation

6.7.2.1 co2

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

Definition at line 231 of file packets.h.

6.7.2.2 generic

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

Definition at line 229 of file packets.h.

6.7.2.3 heartbeat

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

Definition at line 228 of file packets.h.

6.7.2.4 humidity

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

Definition at line 232 of file packets.h.

6.7.2.5 light

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

Definition at line 233 of file packets.h.

6.7.2.6 rgb_light

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

Definition at line 234 of file packets.h.

6.7.2.7 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:

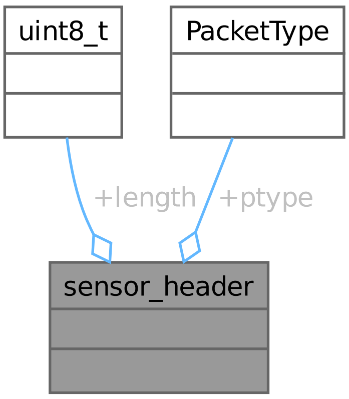
- include/packets.h

6.8 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.).

6.8.1 Detailed Description

Header structure for sensor packets.

Definition at line 40 of file packets.h.

6.8.2 Member Data Documentation

6.8.2.1 length

```
uint8_t sensor_header::length
```

Length of the packet excluding the header.

Definition at line 42 of file packets.h.

6.8.2.2 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:

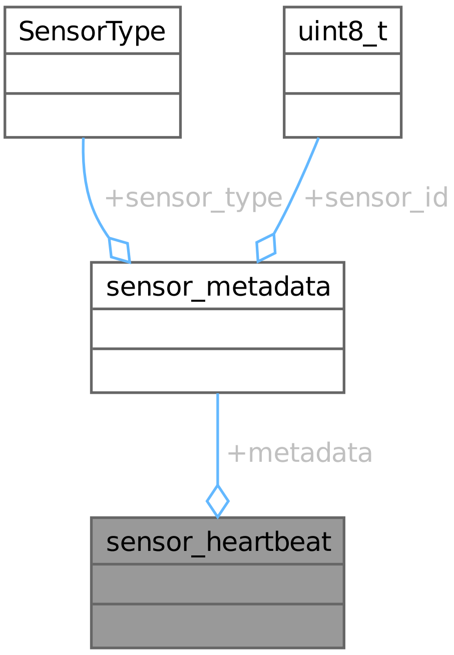
- include/packets.h

6.9 sensor_heartbeat Struct Reference

Structure for heartbeat packets.

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

Collaboration diagram for sensor_heartbeat:



Public Attributes

- struct sensor_metadata metadata

6.9.1 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.

6.9.2 Member Data Documentation

6.9.2.1 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:

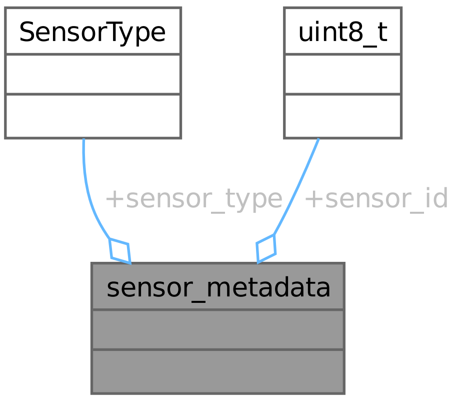
- include/packets.h

6.10 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.

6.10.1 Detailed Description

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

Definition at line 52 of file packets.h.

6.10.2 Member Data Documentation

6.10.2.1 sensor_id

```
uint8_t sensor_metadata::sensor_id
```

ID of the sensor being addressed.

Definition at line 56 of file packets.h.

6.10.2.2 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:

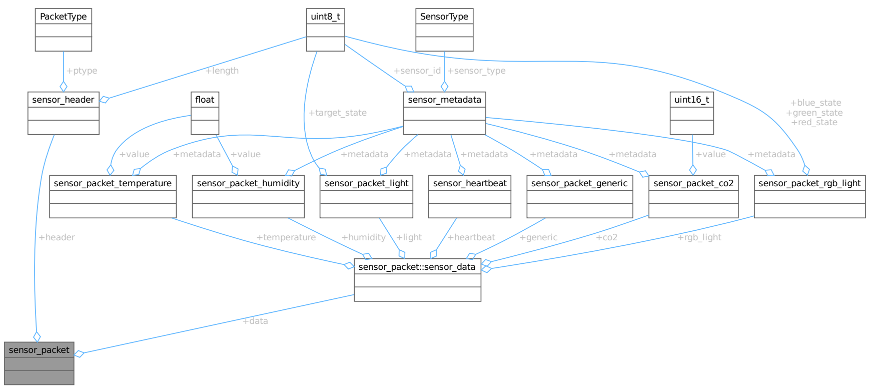
- include/packets.h

6.11 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

6.11.1 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.

6.11.2 Member Data Documentation

6.11.2.1 data

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

6.11.2.2 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:

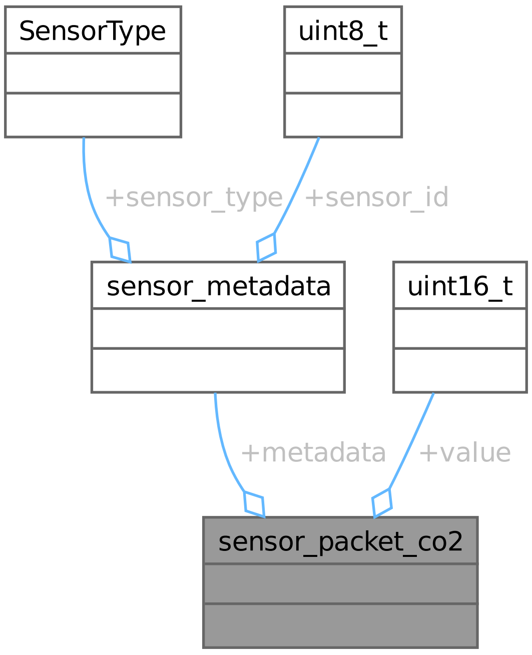
- include/packets.h

6.12 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.

6.12.1 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.

6.12.2 Member Data Documentation

6.12.2.1 metadata

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

Definition at line 108 of file packets.h.

6.12.2.2 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:

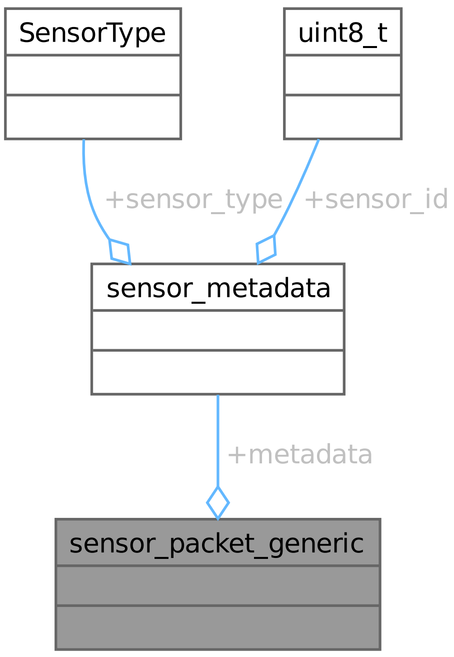
- include/packets.h

6.13 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

6.13.1 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.

6.13.2 Member Data Documentation

6.13.2.1 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:

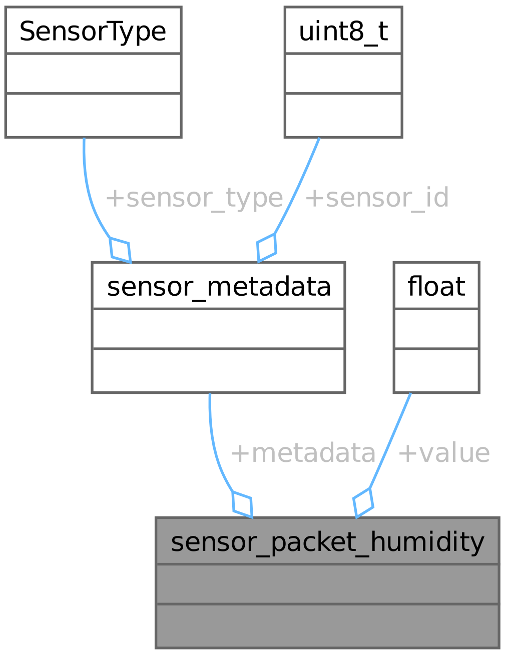
- include/packets.h

6.14 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.

6.14.1 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.

6.14.2 Member Data Documentation

6.14.2.1 metadata

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

Definition at line 121 of file packets.h.

6.14.2.2 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:

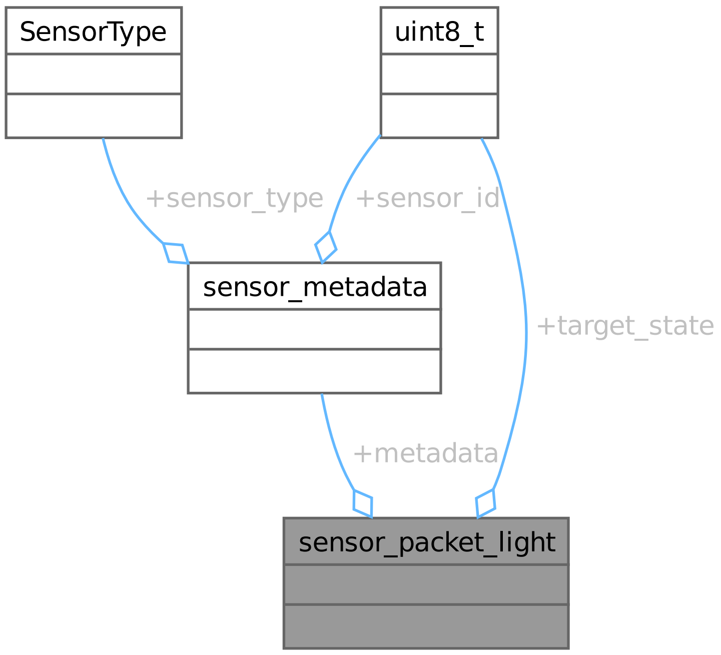
- include/packets.h

6.15 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.

6.15.1 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.

6.15.2 Member Data Documentation

6.15.2.1 metadata

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

Definition at line 134 of file packets.h.

6.15.2.2 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:

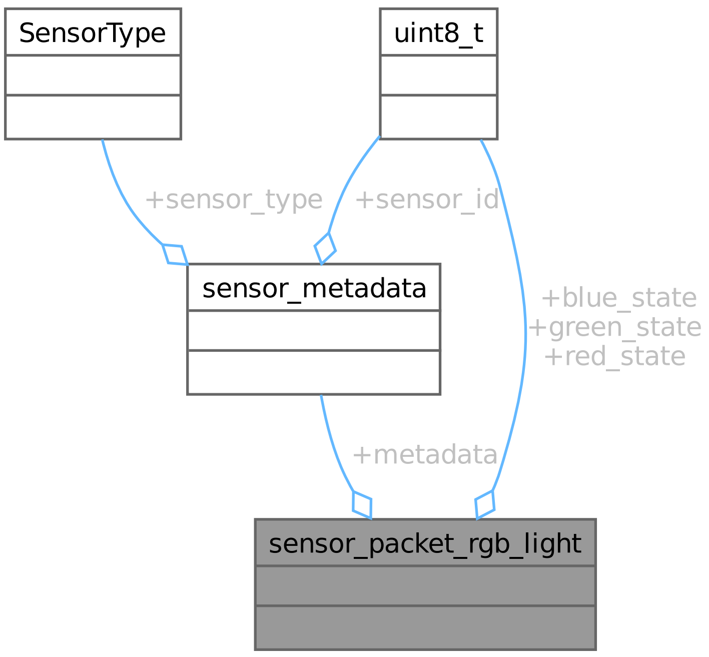
- include/packets.h

6.16 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.

6.16.1 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.

6.16.2 Member Data Documentation

6.16.2.1 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.

6.16.2.2 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.

6.16.2.3 metadata

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

Definition at line 148 of file packets.h.

6.16.2.4 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:

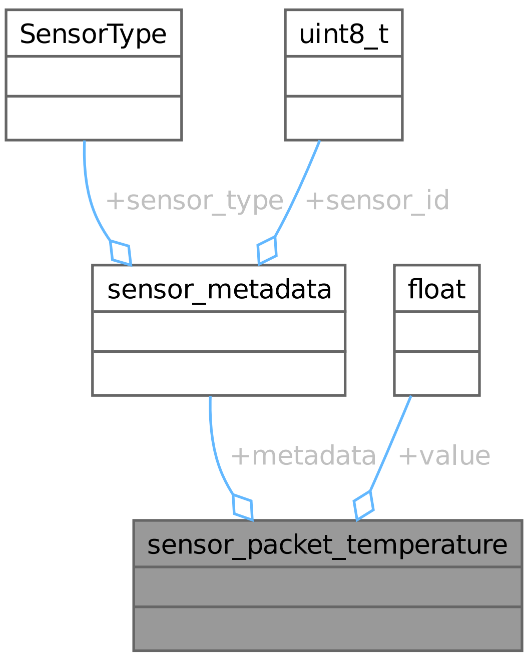
- include/packets.h

6.17 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.

6.17.1 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.

6.17.2 Member Data Documentation

6.17.2.1 metadata

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

Definition at line 95 of file packets.h.

6.17.2.2 value

```
float sensor_packet_temperature::value
```

Value of the sensor reading the temperature represented in Celcius.

Definition at line 97 of file packets.h.

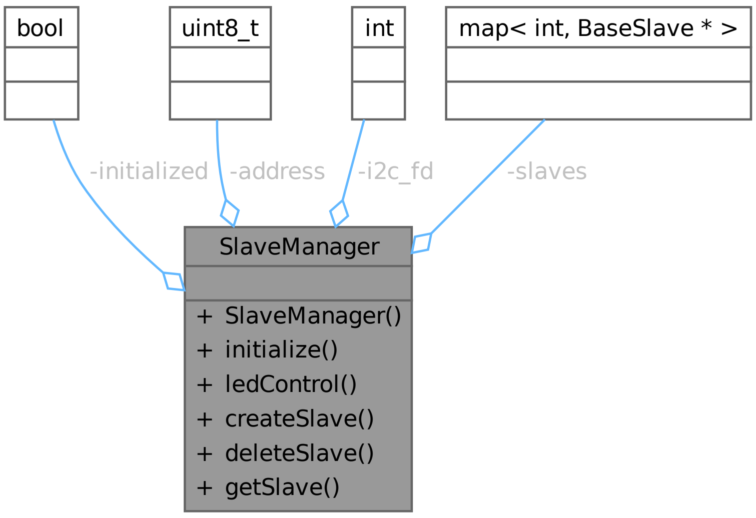
The documentation for this struct was generated from the following file:

- include/packets.h

6.18 SlaveManager Class Reference

```
#include <SlaveManager.h>
```

Collaboration diagram for SlaveManager:



Public Member Functions

- SlaveManager ()
- void initialize ()

  Setup the underlying I2C bus.
- void ledControl (uint8_t led_number, uint8_t led_state)
- void createSlave (SensorType type, int i2c_address)
- void deleteSlave (uint8_t id)

  Unmaps the slave from the internal mapping.
- BaseSlave ∗ getSlave (int id)

  Get a slavedevice with a given ID.

Private Attributes

- bool initialized = false
- uint8_t address = 0x01
- int i2c_fd = -1
- std::map< int, BaseSlave ∗ > slaves

6.18.1 Detailed Description

Definition at line 9 of file SlaveManager.h.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 SlaveManager()

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

Definition at line 15 of file SlaveManager.cpp.

6.18.3 Member Function Documentation

6.18.3.1 createSlave()

```
void SlaveManager::createSlave (
             SensorType type,
             int i2c_address )
```

Definition at line 30 of file SlaveManager.cpp.

6.18.3.2 deleteSlave()

```
void SlaveManager::deleteSlave (
             uint8_t id )
```

Unmaps the slave from the internal mapping.

Parameters

id	The ID of the slave device to unmap

Definition at line 47 of file SlaveManager.cpp.

6.18.3.3 getSlave()

```
BaseSlave * SlaveManager::getSlave (
             int id )
```

Get a slavedevice with a given ID.

Parameters

id	The ID of the slave device to retrieve

Definition at line 56 of file SlaveManager.cpp.

6.18.3.4 initialize()

```
void SlaveManager::initialize ( )
```

Setup the underlying I2C bus.

Exceptions

std__runtime_error	if the I2C setup fails

Definition at line 17 of file SlaveManager.cpp.

6.18.3.5 ledControl()

```
void SlaveManager::ledControl (
             uint8_t led_number,
             uint8_t led_state )
```

Definition at line 28 of file SlaveManager.cpp.

6.18.4 Member Data Documentation

6.18.4.1 address

```
uint8_t SlaveManager::address = 0x01  [private]
```

Definition at line 46 of file SlaveManager.h.

6.18.4.2 i2c_fd

```
int SlaveManager::i2c_fd = -1  [private]
```

Definition at line 47 of file SlaveManager.h.

6.18.4.3 initialized

```
bool SlaveManager::initialized = false  [private]
```

Definition at line 44 of file SlaveManager.h.

6.18.4.4 slaves

```
std::map<int, BaseSlave*> SlaveManager::slaves  [private]
```

Definition at line 49 of file SlaveManager.h.

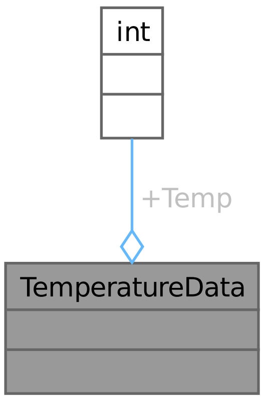
The documentation for this class was generated from the following files:

- include/SlaveManager.h
- src/SlaveManager.cpp

6.19 TemperatureData Struct Reference

```
#include <TemperatureSlave.h>
```

Collaboration diagram for TemperatureData:



Public Attributes

- int Temp

6.19.1 Detailed Description

Definition at line 6 of file TemperatureSlave.h.

6.19.2 Member Data Documentation

6.19.2.1 Temp

```
int TemperatureData::Temp
```

Definition at line 7 of file TemperatureSlave.h.

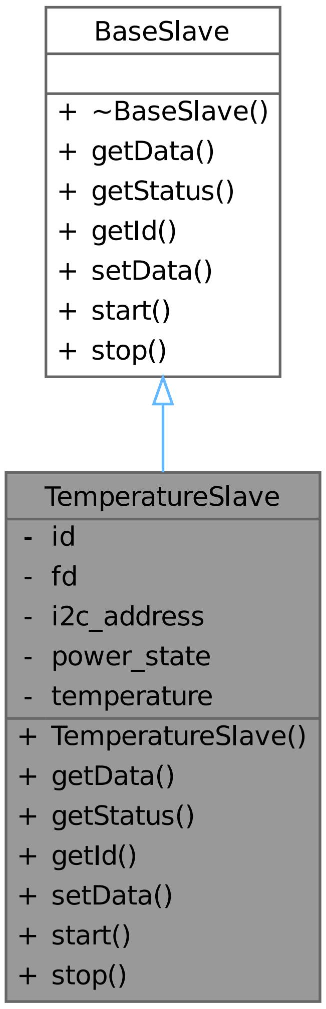
The documentation for this struct was generated from the following file:

- include/TemperatureSlave.h

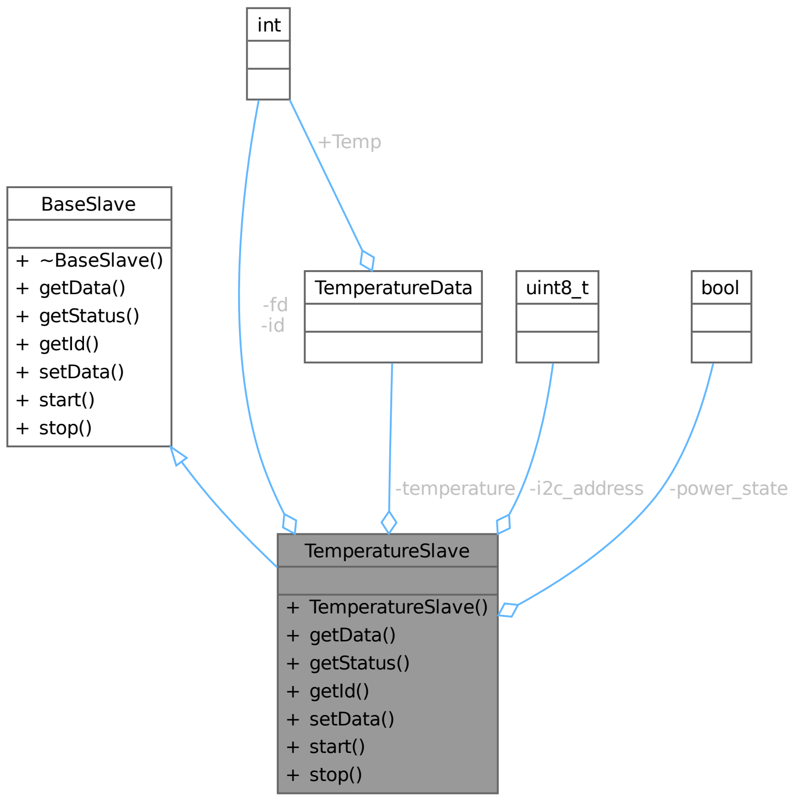
6.20 TemperatureSlave Class Reference

```
#include <TemperatureSlave.h>
```

Inheritance diagram for TemperatureSlave:



Collaboration diagram for TemperatureSlave:



Public Member Functions

- TemperatureSlave (uint8_t id, uint8_t i2c_address)
- void ∗ getData ()
- bool getStatus ()
- int getId ()
- void setData (void ∗data)
- void start (int i2c_fd)
- void stop ()

Public Member Functions inherited from BaseSlave

- virtual ∼BaseSlave ()=default

Private Attributes

- int id
- int fd
- uint8_t i2c_address
- bool power_state
- TemperatureData temperature

6.20.1 Detailed Description

Definition at line 10 of file TemperatureSlave.h.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 TemperatureSlave()

```
TemperatureSlave::TemperatureSlave (
            uint8_t id,
            uint8_t i2c_address )
```

Definition at line 3 of file TemperatureSlave.cpp.

6.20.3 Member Function Documentation

6.20.3.1 getData()

```
void * TemperatureSlave::getData ( )  [virtual]
```

Implements BaseSlave.

Definition at line 6 of file TemperatureSlave.cpp.

6.20.3.2 getId()

```
int TemperatureSlave::getId ( )  [virtual]
```

Implements BaseSlave.

Definition at line 23 of file TemperatureSlave.cpp.

6.20.3.3 getStatus()

```
bool TemperatureSlave::getStatus ( )  [virtual]
```

Implements BaseSlave.

Definition at line 14 of file TemperatureSlave.cpp.

6.20.3.4 setData()

```
void TemperatureSlave::setData (
            void * data )  [virtual]
```

Implements BaseSlave.

Definition at line 25 of file TemperatureSlave.cpp.

6.20.3.5 start()

```
void TemperatureSlave::start (
             int i2c_fd )  [virtual]
```

Implements BaseSlave.

Definition at line 32 of file TemperatureSlave.cpp.

6.20.3.6 stop()

```
void TemperatureSlave::stop ( )  [virtual]
```

Implements BaseSlave.

Definition at line 34 of file TemperatureSlave.cpp.

6.20.4 Member Data Documentation

6.20.4.1 fd

```
int TemperatureSlave::fd  [private]
```

Definition at line 22 of file TemperatureSlave.h.

6.20.4.2 i2c_address

```
uint8_t TemperatureSlave::i2c_address  [private]
```

Definition at line 23 of file TemperatureSlave.h.

6.20.4.3 id

```
int TemperatureSlave::id  [private]
```

Definition at line 21 of file TemperatureSlave.h.

6.20.4.4 power_state

```
bool TemperatureSlave::power_state  [private]
```

Definition at line 25 of file TemperatureSlave.h.

6.20.4.5 temperature

```
TemperatureData TemperatureSlave::temperature  [private]
```

Definition at line 26 of file TemperatureSlave.h.

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

- include/TemperatureSlave.h
- src/TemperatureSlave.cpp

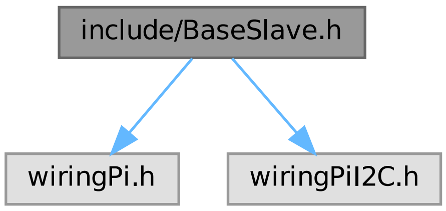
Chapter 7

File Documentation

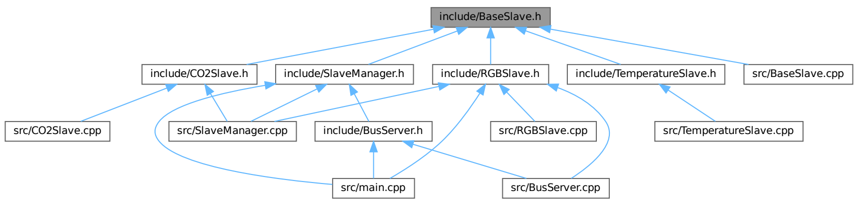
7.1 include/BaseSlave.h File Reference

```
#include <wiringPi.h>
#include <wiringPiI2C.h>
```

Include dependency graph for BaseSlave.h:



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



Classes

- class BaseSlave

7.2 BaseSlave.h

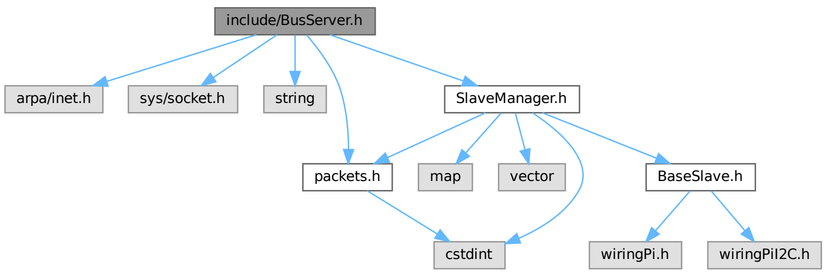
Go to the documentation of this file.

```
#pragma once

#include <wiringPi.h>
#include <wiringPiI2C.h>

class BaseSlave {
   public:
    virtual ~BaseSlave() = default;
    virtual const void* getData() = 0;
    virtual bool getStatus() = 0;
    virtual int getId() = 0;
    virtual void setData(void* data) = 0;
    virtual void start(int i2c_fd) = 0;
    virtual void stop() = 0;
};
```

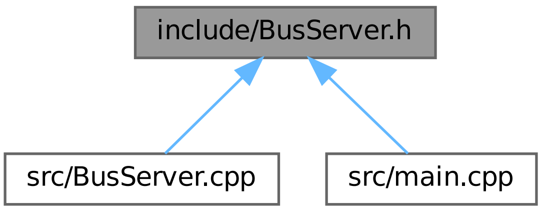
7.3 include/BusServer.h File Reference

```
#include <arpa/inet.h>
#include <sys/socket.h>
#include <string>
#include "SlaveManager.h"
#include "packets.h"
```

Include dependency graph for BusServer.h:



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



Classes

- class BusServer

Macros

- #define BUFFER_SIZE 1024

7.3.1 Macro Definition Documentation

7.3.1.1 BUFFER_SIZE

```
#define BUFFER_SIZE 1024
```

Definition at line 10 of file BusServer.h.

7.4 BusServer.h

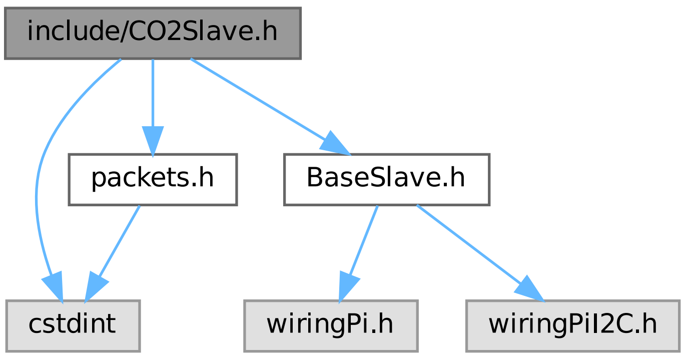
Go to the documentation of this file.

```
00001 #pragma once
00002 #include <arpa/inet.h>
00003 #include <sys/socket.h>
00004 
00005 #include <string>
00006 
00007 #include "SlaveManager.h"
00008 #include "packets.h"
00009 
00010 #define BUFFER_SIZE 1024
00011 
00012 class BusServer {
00013    public:
00014     BusServer() : listening_fd(-1) {}
00015 
00025     void setup(std::string ip, int port);
00026 
00031     void listen();
00032 
00041     void send(struct sensor_packet* pkt, int fd);
00042 
00048     void start();
00049 
00053     SlaveManager& getSlaveManager();
00054 
00055    private:
00056     int listening_fd;
00057 
00058     struct sockaddr_in listening_address;
00059     bool wemos_bridge_connected = false;
00060     char buffer[BUFFER_SIZE];
00061 
00062     SlaveManager slave_manager;
00063 };
```

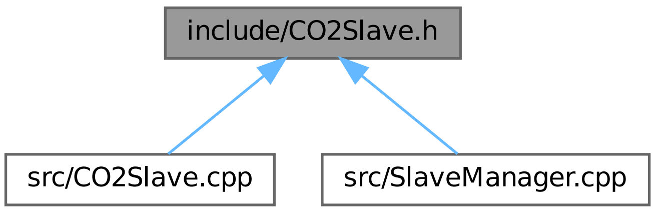
7.5 include/CO2Slave.h File Reference

```
#include <cstdint>
#include "BaseSlave.h"
#include "packets.h"
```

Include dependency graph for CO2Slave.h:



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



Classes

- class CO2Slave

7.6 CO2Slave.h

Go to the documentation of this file.

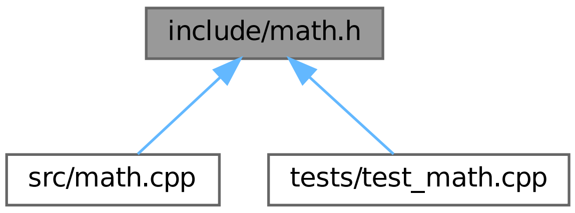
```
#pragma once
#include <cstdint>

#include "BaseSlave.h"
```

```
#include "packets.h"

class CO2Slave : public BaseSlave {
   public:
    CO2Slave(uint8_t id, uint8_t i2c_address);

    const void* getData() override;
    bool getStatus() override;
    int getId() override;
    void setData(void* data) override;
    void start(int i2c_fd) override;
    void stop() override;

   private:
    int id;
    int fd;
    uint8_t i2c_address;
    uint16_t command;

    bool power_state = true;
    struct sensor_packet state_packet;
};
```

7.7 include/math.h File Reference

Header file for math.cpp.

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



Functions

- int add (int a, int b)

  Adds two integers.
- int subtract (int a, int b)

  Subtracts two integers.

7.7.1 Detailed Description

Header file for math.cpp.

This file contains declarations for basic math operations.

Author

Daan Breur

Definition in file math.h.

7.7.2 Function Documentation

7.7.2.1 add()

```
int add (
             int a,
             int b )
```

Adds two integers.

Parameters

a	First integer.
b	Second integer.

Returns

The sum of a and b.

This function takes two integers as input and returns their sum.

Definition at line 16 of file math.cpp.

7.7.2.2 subtract()

```
int subtract (
             int a,
             int b )
```

Subtracts two integers.

Parameters

a	First integer.
b	Second integer.

Returns

The difference of a and b.

This function takes two integers as input and returns the result of subtracting b from a.

Definition at line 26 of file math.cpp.

7.8 math.h

Go to the documentation of this file.

```
00001
```

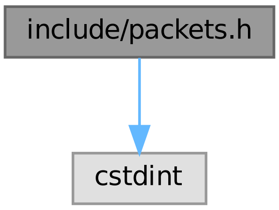
```
#ifndef MATH_H
#define MATH_H

int add(int a, int b);
int subtract(int a, int b);

#endif
```

7.9 include/packets.h File Reference

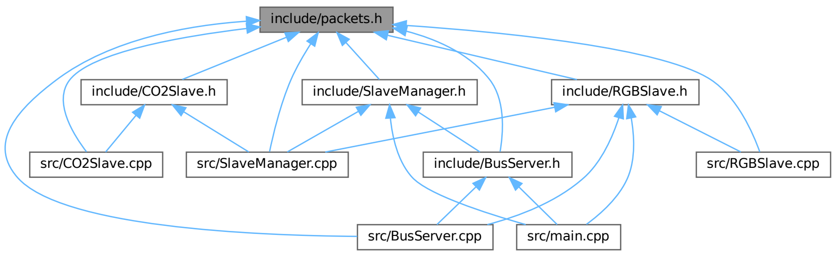
Header file for packets.h.

`#include <cstdint>`

Include dependency graph for packets.h:



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



Classes

- struct sensor_header

  Header structure for sensor packets.
- struct sensor_metadata

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

  Structure for heartbeat packets.
- struct sensor_packet_generic

  Structure for generic sensor packets.

- struct sensor_packet_temperature

  Structure for temperature sensor packets.
- struct sensor_packet_co2

  Structure for CO2 sensor packets.
- struct sensor_packet_humidity

  Structure for humidity sensor packets.
- struct sensor_packet_light

  Structure for light sensor packets.
- struct sensor_packet_rgb_light

  Structure for RGB light sensor packets.
- struct sensor_packet

  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

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

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

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

- uint8_t green_state

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

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

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

7.9.1 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.

7.9.2 Enumeration Type Documentation

7.9.2.1 PacketType

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

Enumerator

DATA	
HEARTBEAT	
DASHBOARD_POST	
DASHBOARD_GET	
DASHBOARD_RESPONSE	

Definition at line 27 of file packets.h.

7.9.2.2 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.

7.9.3 Function Documentation

7.9.3.1 __attribute__()

```
struct sensor_header __attribute__ (
             (packed)   )
```

7.9.4 Variable Documentation

7.9.4.1 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.

7.9.4.2 data

```
union sensor_data data
```

7.9.4.3 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.

7.9.4.4 header

```
struct sensor_header header
```

Header of the packet containing length and type information.

Definition at line 1 of file packets.h.

7.9.4.5 length

```
uint8_t length
```

Length of the packet excluding the header.

Definition at line 1 of file packets.h.

7.9.4.6 metadata

```
struct sensor_metadata metadata
```

Definition at line 0 of file packets.h.

7.9.4.7 ptype

```
PacketType ptype
```

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

Definition at line 3 of file packets.h.

7.9.4.8 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.

7.9.4.9 sensor_id

```
uint8_t sensor_id
```

ID of the sensor being addressed.

Definition at line 3 of file packets.h.

7.9.4.10 sensor_type

```
SensorType sensor_type
```

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

Definition at line 1 of file packets.h.

7.9.4.11 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.

7.9.4.12 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.

7.10 packets.h

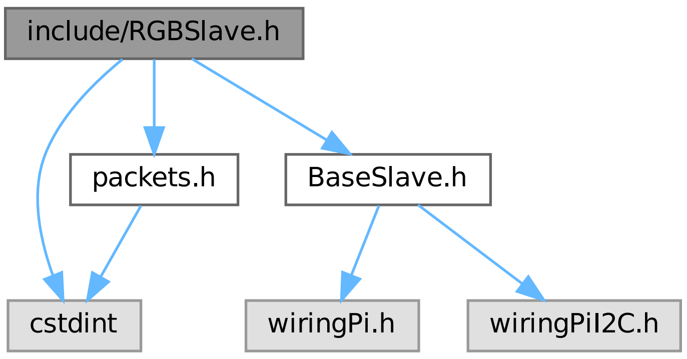
Go to the documentation of this file.

```
00001
00010 #ifndef PACKETS_H
00011 #define PACKETS_H
00012
00013 #include <cstdint>
00014
00015 enum class SensorType : uint8_t {
00016     NOOP = 0,
00017     BUTTON = 1,
00018     TEMPERATURE = 2,
00019     CO2 = 3,
00020     HUMIDITY = 4,
00021     PRESSURE = 5,
00022     LIGHT = 6,
00023     MOTION = 7,
00024     RGB_LIGHT = 8,
00025 };
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 {
    SensorType sensor_type;
    uint8_t sensor_id;
} __attribute__((packed));

// Specific packet structures (ensure alignment/packing matches expected format)

struct sensor_heartbeat {
    struct sensor_metadata metadata;
} __attribute__((packed));

struct sensor_packet_generic {
    struct sensor_metadata metadata;
    // /** @brief Whether the sensor did or did not trigger */
    // bool value;
} __attribute__((packed));

struct sensor_packet_temperature {
    struct sensor_metadata metadata;
    float value;
} __attribute__((packed));

struct sensor_packet_co2 {
    struct sensor_metadata metadata;
    uint16_t value;
} __attribute__((packed));

struct sensor_packet_humidity {
    struct sensor_metadata metadata;
    float value;
} __attribute__((packed));

struct sensor_packet_light {
    struct sensor_metadata metadata;
    uint8_t target_state;
} __attribute__((packed));

struct sensor_packet_rgb_light {
    struct sensor_metadata metadata;
    uint8_t red_state;
    uint8_t green_state;
    uint8_t blue_state;
} __attribute__((packed));
// --- End Structures ---

struct sensor_packet {
    struct sensor_header header;

    union sensor_data {
        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;
    } data;
} __attribute__((packed));

#endif  // PACKETS_H
```

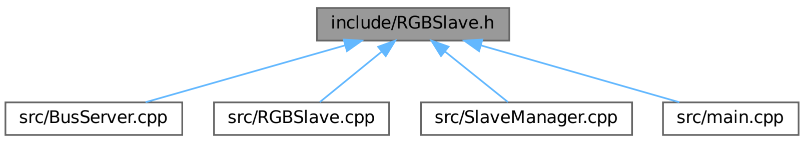
7.11 include/RGBSlave.h File Reference

```
#include <cstdint>
#include "BaseSlave.h"
#include "packets.h"
```

Include dependency graph for RGBSlave.h:



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



Classes

- struct RGBData
- class RGBSlave

Functions

- struct RGBData __attribute__ ((packed))

Variables

- uint8_t R
- uint8_t G
- uint8_t B
- RGBSlave __attribute__

7.11.1 Function Documentation

7.11.1.1 __attribute__()

```
struct RGBData __attribute__ (
              (packed)   )
```

7.11.2 Variable Documentation

7.11.2.1 __attribute__

```
struct sensor_packet __attribute__
```

7.11.2.2 B

```
uint8_t B
```

Definition at line 0 of file RGBSlave.h.

7.11.2.3 G

```
uint8_t G
```

Definition at line 0 of file RGBSlave.h.

7.11.2.4 R

```
uint8_t R
```

Definition at line 0 of file RGBSlave.h.

7.12 RGBSlave.h

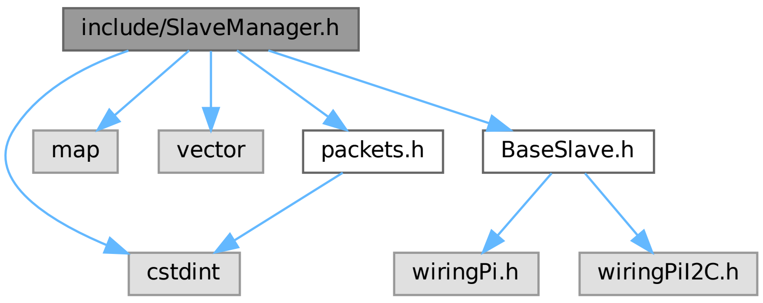
Go to the documentation of this file.

```
#pragma once
#include <cstdint>

#include "BaseSlave.h"
#include "packets.h"

struct RGBData {
    uint8_t R, G, B;
} __attribute__((packed));

class RGBSlave : public BaseSlave {
   public:
    RGBSlave(uint8_t id, uint8_t i2c_address);
    const void* getData();
    bool getStatus();
    int getId();
    void setData(void* data);
    void start(int i2c_fd);
    void stop();

   private:
    int id;
    int fd;
    uint8_t i2c_address;

    RGBData color_state;

    struct sensor_packet state_packet;
};
```

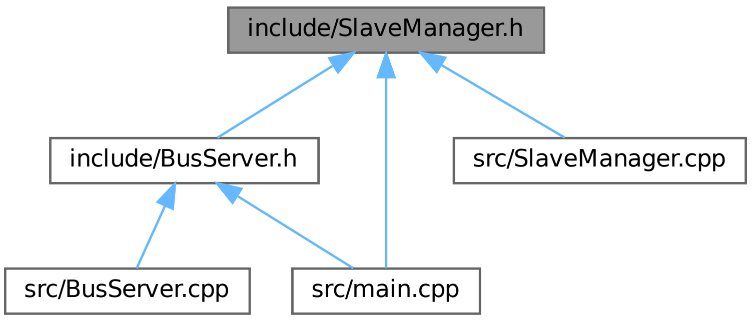
7.13 include/SlaveManager.h File Reference

```
#include <cstdint>
#include <map>
#include <vector>
#include "BaseSlave.h"
#include "packets.h"
```

Include dependency graph for SlaveManager.h:



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



Classes

- class SlaveManager

7.14 SlaveManager.h

Go to the documentation of this file.

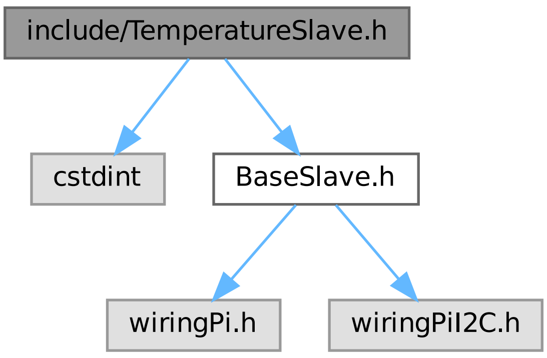
```
00001 #pragma once
```

```
#include <cstdint>
#include <map>
#include <vector>

#include "BaseSlave.h"
#include "packets.h"

class SlaveManager {
  public:
   SlaveManager();

   void initialize();

   void ledControl(uint8_t led_number, uint8_t led_state);

   /*
    * @brief Creates a new slave device into the internal mapping
    * @details Assigns a pre-known SensorType and ID to an I2C device, and then opens the I2C
    * connection to it
    * @param type The SensorType of the sensor, as defined in packets.h
    * @param id The ID of the slave device
    * @param i2c_address The I2C address of the slave device
    */
   void createSlave(SensorType type, int i2c_address);

   void deleteSlave(uint8_t id);

   BaseSlave* getSlave(int id);

  private:
   bool initialized = false;

   uint8_t address = 0x01;
   int i2c_fd = -1;

   std::map<int, BaseSlave*> slaves;  // maps ID to BaseSlave ptr
};
```

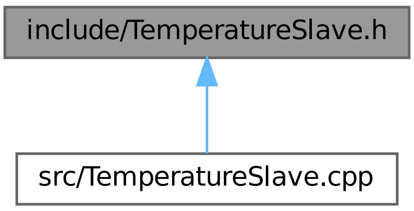
7.15 include/TemperatureSlave.h File Reference

```
#include <cstdint>
#include "BaseSlave.h"
```

Include dependency graph for TemperatureSlave.h:



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



Classes

- struct TemperatureData
- class TemperatureSlave

Functions

- struct TemperatureData __attribute__ ((packed))

Variables

- int Temp
- TemperatureSlave __attribute__

7.15.1 Function Documentation

7.15.1.1 __attribute__()

```
struct TemperatureData __attribute__ (
             (packed)   )
```

7.15.2 Variable Documentation

7.15.2.1 __attribute__

```
TemperatureSlave __attribute__
```

7.15.2.2 Temp

```
int Temp
```

Definition at line 0 of file TemperatureSlave.h.

7.16 TemperatureSlave.h

Go to the documentation of this file.

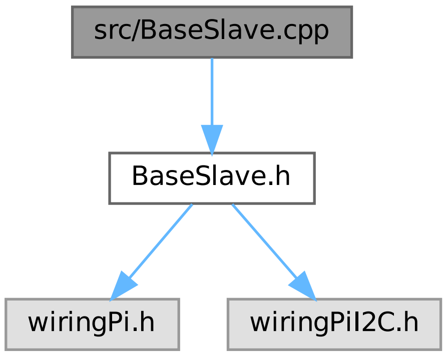
```
#pragma once
#include <cstdint>

#include "BaseSlave.h"

struct TemperatureData {
    int Temp;
} __attribute__((packed));

class TemperatureSlave : public BaseSlave {
   public:
    TemperatureSlave(uint8_t id, uint8_t i2c_address);
    void* getData();
    bool getStatus();
    int getId();
    void setData(void* data);
    void start(int i2c_fd);
    void stop();

   private:
    int id;
    int fd;
    uint8_t i2c_address;

    bool power_state;
    TemperatureData temperature;
};
```

7.17 README.md File Reference

7.18 src/BaseSlave.cpp File Reference

`#include "BaseSlave.h"`

Include dependency graph for BaseSlave.cpp:



7.19 BaseSlave.cpp

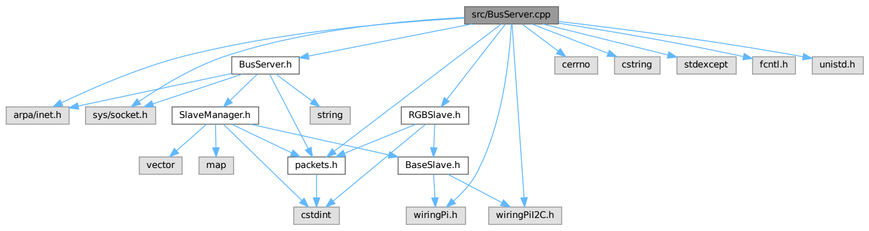
Go to the documentation of this file.

```
#include "BaseSlave.h"
```

7.20 src/BusServer.cpp File Reference

```
#include "BusServer.h"
#include <arpa/inet.h>
#include <sys/socket.h>
#include <cerrno>
#include <cstring>
#include <stdexcept>
#include <fcntl.h>
#include <unistd.h>
#include "RGBSlave.h"
#include "packets.h"
#include "wiringPi.h"
#include "wiringPiI2C.h"
```

Include dependency graph for BusServer.cpp:



Macros

- #define RGB_SLAVE_ADDRESS 0x69

7.20.1 Macro Definition Documentation

7.20.1.1 RGB_SLAVE_ADDRESS

```
#define RGB_SLAVE_ADDRESS 0x69
```

Definition at line 18 of file BusServer.cpp.

7.21 BusServer.cpp

Go to the documentation of this file.

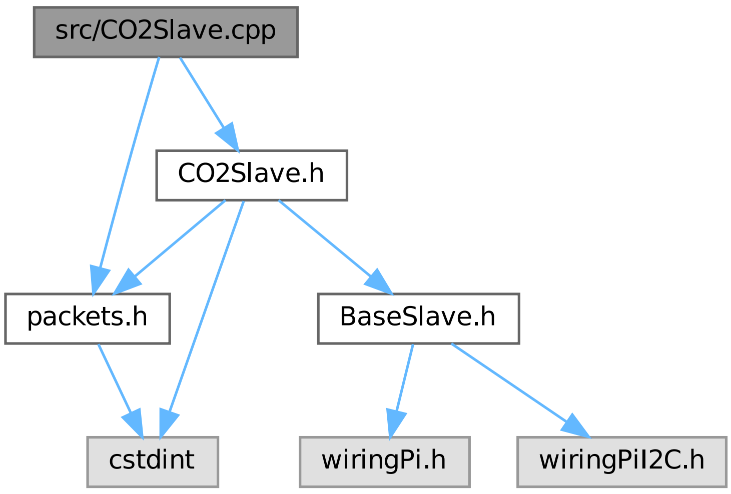
```
00001 #include "BusServer.h"
00002
00003 #include <arpa/inet.h>
00004 #include <sys/socket.h>
00005
00006 #include <cerrno>
00007 #include <cstring>
00008 #include <stdexcept>
00009
00010 #include <fcntl.h>
00011 #include <unistd.h>
00012
00013 #include "RGBSlave.h"
00014 #include "packets.h"
```

```
#include "wiringPi.h"
#include "wiringPiI2C.h"

#define RGB_SLAVE_ADDRESS 0x69

void BusServer::setup(std::string ip, int port) {
    // slave_manager.initialize();

    memset(&listening_address, 0, sizeof(listening_address));

    if (port <= 0 || port > 65535) {
        throw std::invalid_argument("invalid port passed");
    }

    if (0 == inet_aton(ip.c_str(), &listening_address.sin_addr)) {
        perror("inet_aton()");
        throw std::invalid_argument("invalid IP address passed");
    }

    listening_address.sin_family = AF_INET;
    listening_address.sin_port = htons(port);

    listening_fd = socket(AF_INET, SOCK_STREAM | SOCK_NONBLOCK, 0);
    if (-1 == listening_fd) {
        perror("socket()");
        throw std::runtime_error("failed to create socket");
    }

    {
        int one = 1;
        if (-1 ==
            setsockopt(listening_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &one, sizeof(one))) {
            perror("setsockopt()");
            throw std::runtime_error("failed to set socket options");
        }
    }

    socklen_t len = sizeof(listening_address);

    if (-1 == bind(listening_fd, (struct sockaddr*)&listening_address, len)) {
        perror("bind()");
        throw std::runtime_error("failed to bind socket");
    }
}

void BusServer::listen() {
    if (-1 == ::listen(listening_fd, 8)) {
        perror("listen()");
        throw std::runtime_error("failed to listen on socket");
    }
}

void BusServer::send(struct sensor_packet* packet_ptr, int fd) {
    if (!wemos_bridge_connected) {
        throw std::runtime_error("no longer connected to socket");
    }
    if (-1 == ::send(fd, packet_ptr, sizeof(struct sensor_header) + packet_ptr->header.length, 0)) {
        perror("send()");
        throw std::runtime_error("failed to send on socket");
    }
}

void BusServer::start() {
    // start off by mapping pre-known addresses to ID's
    listen();

    int the_fd = -1;

    while (true) {
        char buffer[32] = {0};
        bool net_request = false;

        int new_fd = -1;

        {
            struct sockaddr_in client_address = {0};
            socklen_t client_address_length = sizeof(client_address);

        new_fd = accept4(listening_fd, (struct sockaddr*)&client_address,
                         &client_address_length, SOCK_NONBLOCK);


            if (-1 == new_fd) {

                int err = errno;
                switch (err) {
                    case EWOULDBLOCK:
                        // no client has tried to connect; no big deal
                        break;

                    default:
                        perror("accept4()");
                        throw std::runtime_error("accept4() failed");
                        break;
                }
            } else {
        the_fd = new_fd;
            //printf("new fd: %d\nthe fd: %d\n", new_fd, the_fd);
        }


        if (the_fd != -1) {
            int err;
        int recvd = recv(the_fd, buffer, sizeof(buffer), SOCK_NONBLOCK);
        if (0 == recvd) {
            close(the_fd);
            the_fd = -1;
            continue;
        }

        if (-1 == recvd)
            err = errno;
            //printf("%d\n", err);
            //perror("recv balls");
                if (-1 == recvd && err != EAGAIN) {
                    perror("recv()");
            usleep(100000);
                    throw std::runtime_error("reading from client socket failed for some reason");
                } else if (errno != EAGAIN) {
            //printf("%d\n", recvd);
                } else {
            // printf("%d\n", recvd);
            if (recvd > 0)
                net_request = true;

                    //perror("recv()[1]");
            usleep(100000);
        }
            }
        }

        if (net_request) {
            struct sensor_packet* pkt_ptr = (struct sensor_packet*)buffer;
            BaseSlave* the_slave = slave_manager.getSlave(pkt_ptr->data.generic.metadata.sensor_id);
            SensorType the_sensor_type = pkt_ptr->data.generic.metadata.sensor_type;
            uint8_t values[8] = {0};

            switch (pkt_ptr->header.ptype) {
                case PacketType::DASHBOARD_POST:
                    switch (the_sensor_type) {
                        case SensorType::LIGHT:
                            values[0] = pkt_ptr->data.light.target_state;

                            the_slave->setData(values);
                            break;

                        case SensorType::RGB_LIGHT:
                            struct RGBData rgb_data = {.R = pkt_ptr->data.rgb_light.red_state,
                                                       .G = pkt_ptr->data.rgb_light.green_state,
                                                       .B = pkt_ptr->data.rgb_light.blue_state};

                            the_slave->setData(&rgb_data);

                            break;
                    }
                    break;

                case PacketType::DASHBOARD_GET:
                    struct sensor_packet state_ptr;
                    memcpy(&state_ptr, the_slave->getData(), sizeof(struct sensor_packet));

                    state_ptr.header.ptype = PacketType::DASHBOARD_RESPONSE;

                    send(&state_ptr, new_fd);
                    break;
            }
        }
    }
}

SlaveManager& BusServer::getSlaveManager() {
    return slave_manager;
}
```

7.22 src/CO2Slave.cpp File Reference

```
#include "CO2Slave.h"
#include "packets.h"
```

Include dependency graph for CO2Slave.cpp:



7.23 CO2Slave.cpp

Go to the documentation of this file.

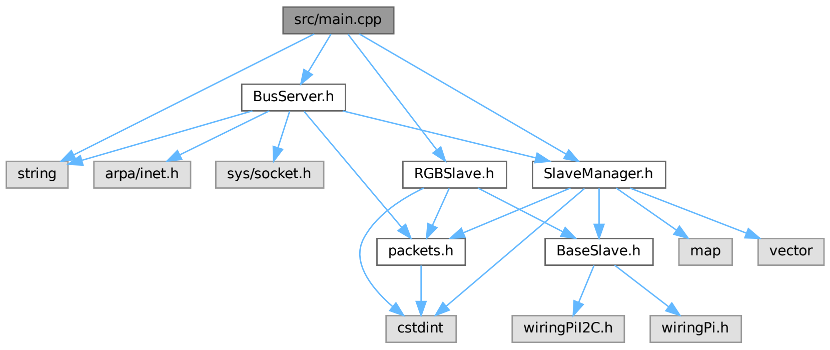
```
#include "CO2Slave.h"

#include "packets.h"

CO2Slave::CO2Slave(uint8_t id, uint8_t i2c_address) : id(id) {
    state_packet.header.ptype = PacketType::DATA;
    state_packet.header.length = sizeof(struct sensor_packet_co2);

    state_packet.data.rgb_light.metadata.sensor_id = id;
    state_packet.data.rgb_light.metadata.sensor_type = SensorType::CO2;
    command = 0x2003;
    uint8_t* command_ptr = (uint8_t*)&command;

    wiringPiI2CWrite(fd, command_ptr[0]);
    wiringPiI2CWrite(fd, command_ptr[1]);
}

const void* CO2Slave::getData() {
    command = 0x2008;
    uint8_t* command_ptr = (uint8_t*)&command;
    wiringPiI2CWrite(fd, command_ptr[0]);
    wiringPiI2CWrite(fd, command_ptr[1]);
    uint8_t* value_ptr = (uint8_t*)&state_packet.data.co2.value;

    value_ptr[1] = wiringPiI2CRead(fd);
    value_ptr[0] = wiringPiI2CRead(fd);
    uint8_t empty = wiringPiI2CRead(fd);
    return &state_packet;
}

bool CO2Slave::getStatus() { return power_state; }

int CO2Slave::getId() { return id; }
```

```
void CO2Slave::setData(void* data) { return; }

void CO2Slave::start(int i2c_fd) { fd = i2c_fd; }

void CO2Slave::stop() { fd = -1; }
```

7.24 src/main.cpp File Reference

```
#include <string>
#include "BusServer.h"
#include "RGBSlave.h"
#include "SlaveManager.h"
```

Include dependency graph for main.cpp:



Functions

- int main ()

7.24.1 Function Documentation

7.24.1.1 main()

```
int main ( )
```

Definition at line 7 of file main.cpp.

7.25 main.cpp

Go to the documentation of this file.

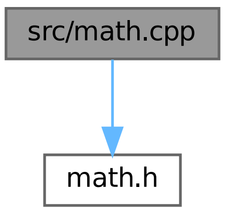
```
#include <string>

#include "BusServer.h"
#include "RGBSlave.h"
#include "SlaveManager.h"

int main() {
    BusServer bus_server;
    bus_server.setup("0.0.0.0", 5000);

    SlaveManager& slave_manager_ref = bus_server.getSlaveManager();

    {
    int fd = wiringPiI2CSetup(0x69);

    slave_manager_ref.createSlave(SensorType::RGB_LIGHT, 0x69);
    slave_manager_ref.getSlave(0x69)->start(fd);
    }

    bus_server.start();
}
```

7.26 src/math.cpp File Reference

Implementation of basic math operations.

```
#include "math.h"
```

Include dependency graph for math.cpp:



Functions

- int add (int a, int b)

  Adds two integers.
- int subtract (int a, int b)

  Subtracts two integers.

7.26.1 Detailed Description

Implementation of basic math operations.

Author

Daan Breur

Definition in file math.cpp.

7.26.2 Function Documentation

7.26.2.1 add()

```
int add (
            int a,
            int b )
```

Adds two integers.

Parameters

a	First integer.
b	Second integer.

Returns

The sum of a and b.

This function takes two integers as input and returns their sum.

Definition at line 16 of file math.cpp.

7.26.2.2 subtract()

```
int subtract (
            int a,
            int b )
```

Subtracts two integers.

Parameters

a	First integer.
b	Second integer.

Returns

The difference of a and b.

This function takes two integers as input and returns the result of subtracting b from a.

Definition at line 26 of file math.cpp.

7.27 math.cpp

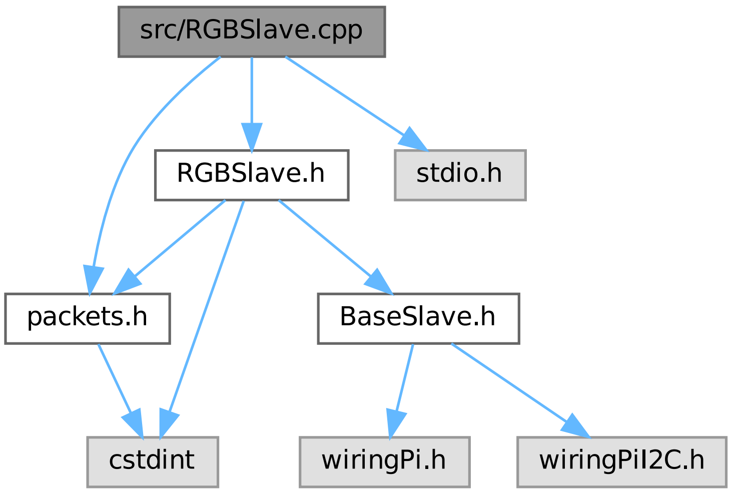
Go to the documentation of this file.

```
00001
00006 #include "math.h"
00007
00016 int add(int a, int b) { return a + b; }
00017
00026 int subtract(int a, int b) { return a - b; }
```

7.28 src/RGBSlave.cpp File Reference

```
#include "RGBSlave.h"
#include <stdio.h>
```

`#include "packets.h"`
Include dependency graph for RGBSlave.cpp:



7.29 RGBSlave.cpp

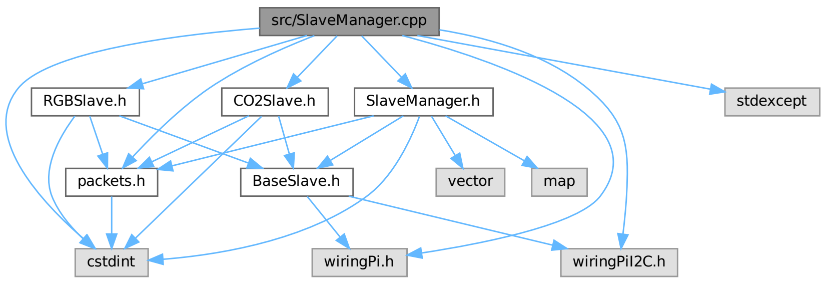
Go to the documentation of this file.

```
#include "RGBSlave.h"

#include <stdio.h>

#include "packets.h"

RGBSlave::RGBSlave(uint8_t id, uint8_t i2c_address) : id(id), i2c_address(i2c_address) {
    state_packet.header.ptype = PacketType::DATA;
    state_packet.header.length = sizeof(struct sensor_packet_rgb_light);

    state_packet.data.rgb_light.metadata.sensor_id = id;
    state_packet.data.rgb_light.metadata.sensor_type = SensorType::RGB_LIGHT;
}

const void* RGBSlave::getData() {
    uint8_t r, g, b;

    r = wiringPiI2CRead(fd);
    g = wiringPiI2CRead(fd);
    b = wiringPiI2CRead(fd);

    color_state.R = r;
    color_state.G = g;
    color_state.B = b;

    state_packet.data.rgb_light.red_state = r;
    state_packet.data.rgb_light.green_state = g;
    state_packet.data.rgb_light.blue_state = b;

    return &state_packet;
}

bool RGBSlave::getStatus() {
    getData();
    if (color_state.R == 0 && color_state.G == 0 && color_state.B == 0) {
        return false;
    } else {
        return true;
    }
}

int RGBSlave::getId() { return id; }

void RGBSlave::setData(void* data) {
    RGBData* rgb_data = (RGBData*)data;
    uint8_t r = rgb_data->R;
    uint8_t g = rgb_data->G;
    uint8_t b = rgb_data->B;

    char command[256] = { 0 };
    snprintf(command, sizeof(command) - 1, "/usr/sbin/i2cset -y 1 0x%02x 0x%02x 0x%02x 0x%02x i", id,
 r, g, b);

    popen(command, "r");

    // wiringPiI2CWriteBlockData(fd, 1, (uint8_t*)rgb_data, 3);
    // wiringPiI2CWrite(fd, g);
    // wiringPiI2CWrite(fd, b);
}

void RGBSlave::start(int i2c_fd) { fd = i2c_fd; }

void RGBSlave::stop() { fd = -1; }
```

7.30 src/SlaveManager.cpp File Reference

```
#include "SlaveManager.h"
#include <wiringPi.h>
#include <wiringPiI2C.h>
#include <cstdint>
#include <stdexcept>
#include "CO2Slave.h"
#include "RGBSlave.h"
#include "packets.h"
```

Include dependency graph for SlaveManager.cpp:



Macros

- #define MASTER_ADDRESS 0x01

7.30.1 Macro Definition Documentation

7.30.1.1 MASTER_ADDRESS

```
#define MASTER_ADDRESS 0x01
```

Definition at line 13 of file SlaveManager.cpp.

7.31 SlaveManager.cpp

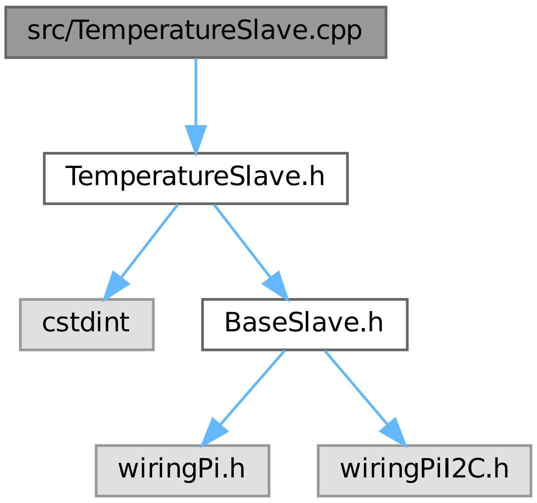
Go to the documentation of this file.

```
#include "SlaveManager.h"

#include <wiringPi.h>
#include <wiringPiI2C.h>

#include <cstdint>
#include <stdexcept>

#include "CO2Slave.h"
#include "RGBSlave.h"
#include "packets.h"

#define MASTER_ADDRESS 0x01

SlaveManager::SlaveManager() : address(MASTER_ADDRESS) {}

void SlaveManager::initialize() {
    // maybe need setup not sure.

    i2c_fd = wiringPiI2CSetup(address);
    if (-1 == i2c_fd) {
        throw std::runtime_error("I2C setup failed");
    } else {
        printf("I2C setup succesful");
    };
}

void SlaveManager::ledControl(uint8_t led_number, uint8_t led_state) {}

void SlaveManager::createSlave(SensorType type, int i2c_address) {
    BaseSlave* newSlave = nullptr;
    switch (type) {
        case SensorType::CO2:
            newSlave = new CO2Slave(i2c_address, i2c_address);
            break;
        case SensorType::RGB_LIGHT:
            newSlave = new RGBSlave(i2c_address, i2c_address);
            break;
        // case /* door */:
        //     newSlave = new DoorSlave(id, i2c_address);
        //     break;
        default:
            return;  // Invalid type
    }
    slaves[i2c_address] = newSlave;
}
void SlaveManager::deleteSlave(uint8_t id) {
    if (nullptr != slaves[id]) {
        /*
         * Disconnect I2C device from bus
         */
        delete slaves[id];
    }
}

BaseSlave* SlaveManager::getSlave(int id) { return slaves[id]; }
```

7.32 src/TemperatureSlave.cpp File Reference

`#include "TemperatureSlave.h"`

Include dependency graph for TemperatureSlave.cpp:



7.33 TemperatureSlave.cpp

Go to the documentation of this file.

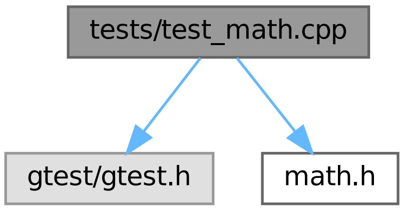
```
#include "TemperatureSlave.h"

TemperatureSlave::TemperatureSlave(uint8_t id, uint8_t i2c_address)
    : id(id), i2c_address(i2c_address) {}

void* TemperatureSlave::getData() {
    int temp;

    temp = wiringPiI2CRead(fd);

    return;
}

bool TemperatureSlave::getStatus() {
    getData();
    if (temperature.Temp == 0) {
        return false;
    } else {
        return true;
    }
}

int TemperatureSlave::getId() { return id; }

void TemperatureSlave::setData(void* data) {
    TemperatureData* temp_data = (TemperatureData*)data;
    int temp = temp_data->Temp;

    wiringPiI2CWrite(fd, temp);
}

void TemperatureSlave::start(int i2c_fd) { fd = i2c_fd; }

void TemperatureSlave::stop() { fd = -1; }
```

7.34 tests/test_math.cpp File Reference

Unit tests for mathematical operations using Google Test framework.

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

Include dependency graph for test_math.cpp:



Functions

- TEST (MathTest, Add)
- TEST (MathTest, Subtract)
- TEST (MathTest, SubtractNegative)

7.34.1 Detailed Description

Unit tests for mathematical operations using Google Test framework.

This file contains test cases for verifying the correctness of functions defined in the "math.h" header. The tests ensure that the mathematical operations behave as expected under various conditions.

Test MathTest.Add

- Verifies that the `add` function correctly computes the sum of two integers.
- Example: `add(2, 3)` should return `5`.

Test MathTest.Subtract

- Verifies that the `subtract` function correctly computes the difference between two integers.
- Examples:
  - `subtract(10, 3)` should return `7`.
  - `subtract(9, 3)` should return `6`.

Test MathTest.SubtractNegative

- Verifies that the `subtract` function handles subtraction with negative integers correctly.
- Example: `subtract(10, -3)` should return `13`.

Definition in file test_math.cpp.

7.34.2 Function Documentation

7.34.2.1 TEST() [1/3]

```
TEST (
            MathTest ,
            Add  )
```

Definition at line 29 of file test_math.cpp.

7.34.2.2 TEST() [2/3]

```
TEST (
            MathTest ,
            Subtract  )
```

Definition at line 31 of file test_math.cpp.

7.34.2.3 TEST() [3/3]

```
TEST (
            MathTest ,
            SubtractNegative  )
```

Definition at line 36 of file test_math.cpp.

7.35 test_math.cpp

Go to the documentation of this file.

```
00001
00025 #include <gtest/gtest.h>
00026
00027 #include "math.h"
00028
00029 TEST(MathTest, Add) { EXPECT_EQ(add(2, 3), 5); }
00030
00031 TEST(MathTest, Subtract) {
00032     EXPECT_EQ(subtract(10, 3), 7);
00033     EXPECT_EQ(subtract(9, 3), 6);
00034 }
00035
00036 TEST(MathTest, SubtractNegative) { EXPECT_EQ(subtract(10, -3), 13); }
```

Index