Embedded C/C++ Final Project - Master Raspberry Pi

1.0

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

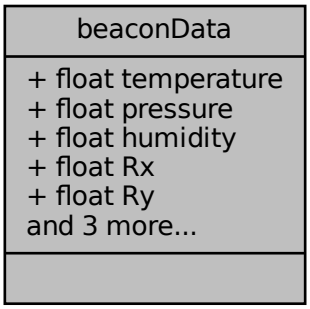
Class Documentation

3.1 beaconData Struct Reference

The beaconData struct contains data received from the slave device, such as temperature, pressure, humidity, x, y, z axes positions, detection status and a timestamp.

`#include <raspberrymaster.hpp>`

Collaboration diagram for beaconData:



Public Attributes

- float temperature
- float pressure
- float humidity
- float Rx
- float Ry
- float Rz
- int detect
- time_t timestamp

3.1.1 Detailed Description

The beaconData struct contains data received from the slave device, such as temperature, pressure, humidity, x, y, z axes positions, detection status and a timestamp.

Definition at line 33 of file raspberrymaster.hpp.

3.1.2 Member Data Documentation

3.1.2.1 detect

```
int beaconData::detect
```

Detection status

Definition at line 47 of file raspberrymaster.hpp.

3.1.2.2 humidity

```
float beaconData::humidity
```

Humidity in percentage

Definition at line 39 of file raspberrymaster.hpp.

3.1.2.3 pressure

```
float beaconData::pressure
```

Tressure in pascals

Definition at line 37 of file raspberrymaster.hpp.

3.1.2.4 Rx

```
float beaconData::Rx
```

X axis rotation

Definition at line 41 of file raspberrymaster.hpp.

3.1.2.5 Ry

```
float beaconData::Ry
```

Y axis rotation

Definition at line 43 of file raspberrymaster.hpp.

3.1.2.6 Rz

```
float beaconData::Rz
```

Z axis rotation

Definition at line 45 of file raspberrymaster.hpp.

3.1.2.7 temperature

```
float beaconData::temperature
```

Temperature in degree celsius

Definition at line 35 of file raspberrymaster.hpp.

3.1.2.8 timestamp

```
time_t beaconData::timestamp
```

Timestamp

Definition at line 49 of file raspberrymaster.hpp.

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

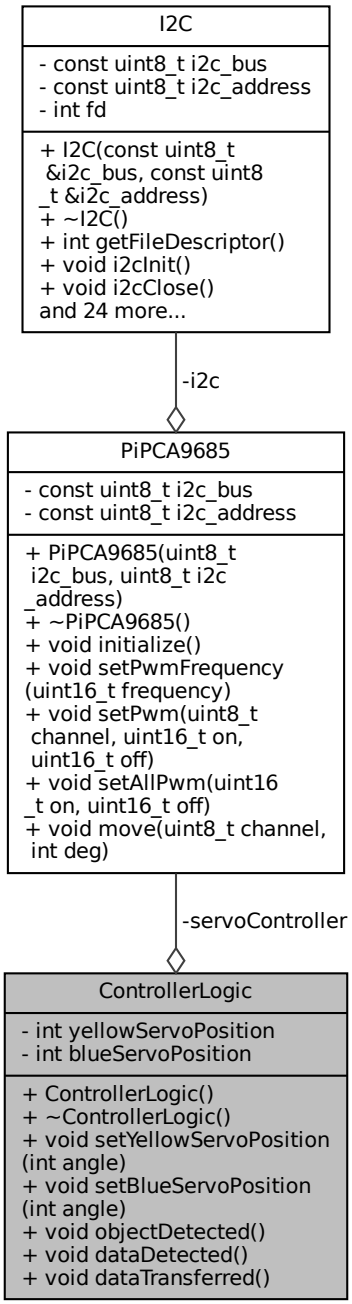
- raspberrymaster.hpp

3.2 ControllerLogic Class Reference

Class for describing the actions that are to be performed by the PCA9685 controller.

`#include <controllerlogic.hpp>`

Collaboration diagram for ControllerLogic:



Public Member Functions

- ControllerLogic ()

  Default constructor for the ControllerLogic class.
- ~ControllerLogic ()

  Default destructor for the ControllerLogic class.
- void setYellowServoPosition (int angle)

  Method to set the angle of the yellow servomotor.
- void setBlueServoPosition (int angle)

  Method to set the angle of the blue servomotor.
- void objectDetected ()

  Method to control the movement of the yellow servomotor when an object is detected.
- void dataDetected ()

  Method to control the movement of the blue servomotor when data is detected.
- void dataTransferred ()

  Method to control the movement of the blue servomotor when data is transferred.

Private Attributes

- PiPCA9685 * servoController

  PiPCA9685 object.
- int yellowServoPosition = 0

  Current position of the yellow flag initialized to 0 degrees.
- int blueServoPosition = 180

  Current position of the blue flag initialized to 180 degrees.

3.2.1 Detailed Description

Class for describing the actions that are to be performed by the PCA9685 controller.

This class describes the different actions which are to be performed by the PCA9685 controller. It allows to control the positions of the yellow and blue flags using servomotors.

Definition at line 27 of file controllerlogic.hpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 ControllerLogic()

```
ControllerLogic::ControllerLogic ( )
```

Default constructor for the ControllerLogic class.

This constructor creates an instance of the PiPCA9685 class, initializes the servomotors, and sets the initial positions of the servomotors.

3.2.2.2 ~ControllerLogic()

```
ControllerLogic::~ControllerLogic ( )
```

Default destructor for the ControllerLogic class.

This destructor sets the servomotors to the initial position before destroying the object.

3.2.3 Member Function Documentation

3.2.3.1 dataDetected()

```
void ControllerLogic::dataDetected ( )
```

Method to control the movement of the blue servomotor when data is detected.

3.2.3.2 dataTransferred()

```
void ControllerLogic::dataTransferred ( )
```

Method to control the movement of the blue servomotor when data is transferred.

3.2.3.3 objectDetected()

```
void ControllerLogic::objectDetected ( )
```

Method to control the movement of the yellow servomotor when an object is detected.

3.2.3.4 setBlueServoPosition()

```
void ControllerLogic::setBlueServoPosition (
            int angle )
```

Method to set the angle of the blue servomotor.

Parameters

angle	The angle to set the servomotor to.

3.2.3.5 setYellowServoPosition()

```
void ControllerLogic::setYellowServoPosition (
            int angle )
```

Method to set the angle of the yellow servomotor.

Parameters

angle	The angle to set the servomotor to

3.2.4 Member Data Documentation

3.2.4.1 blueServoPosition

```
int ControllerLogic::blueServoPosition = 180  [private]
```

Current position of the blue flag initialized to 180 degrees.

Definition at line 90 of file controllerlogic.hpp.

3.2.4.2 servoController

```
PiPCA9685* ControllerLogic::servoController  [private]
```

PiPCA9685 object.

Definition at line 80 of file controllerlogic.hpp.

3.2.4.3 yellowServoPosition

```
int ControllerLogic::yellowServoPosition = 0  [private]
```

Current position of the yellow flag initialized to 0 degrees.

Definition at line 85 of file controllerlogic.hpp.

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

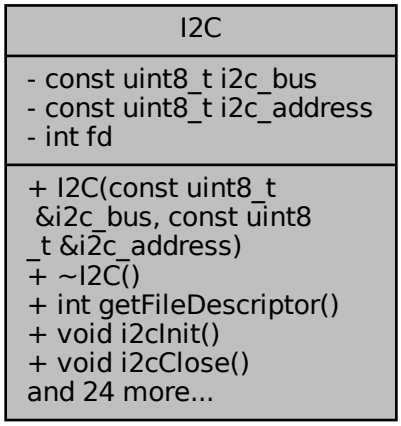
- controllerlogic.hpp

3.3 I2C Class Reference

Class for controlling the I2C bus.

`#include <i2c.hpp>`

Collaboration diagram for I2C:



Public Member Functions

- I2C (const uint8_t &i2c_bus, const uint8_t &i2c_address)
  Constructor for the QI2C class.
- ~I2C ()
  Destructor for the QI2C class.
- int getFileDescriptor ()
  Gets the file descriptor of the I2C device.
- void i2cInit ()
  Initializes the I2C bus.
- void i2cClose ()
  Closes the I2C bus.
- uint8_t readBit8 (uint8_t registerAddress, uint8_t bitNumber)
  Read bit from 8 bit register.
- uint8_t readBit16 (uint16_t registerAddress, uint8_t bitNumber)
  Read bit from 16 bit register.
- uint8_t readBits8 (uint8_t registerAddress, uint8_t bitStart, uint8_t length)
  Read consecutive bits from 8 bit register.
- uint8_t readBits16 (uint16_t registerAddress, uint8_t bitStart, uint8_t length)
  Read consecutive bits from 16 bit register.
- uint8_t readByte8 (uint8_t registerAddress)
  Read a byte from 8 bit register.
- uint8_t readByte16 (uint16_t registerAddress)

Read a byte from 16 bit register.

- void readBytes8 (uint8_t registerAddress, uint8_t length, uint8_t *data)

  Read consecutive bytes from 8 bit register.
- void readBytes16 (uint16_t registerAddress, uint8_t length, uint8_t *data)

  Read consecutive bytes from 16 bit register.
- uint16_t readWord8 (uint8_t registerAddress)

  Read 2 bytes as a word from 8 bit register.
- uint16_t readWord16 (uint16_t registerAddress)

  Read 2 bytes as a word from 16 bit register.
- uint16_t readWord8_2c (uint8_t registerAddress)

  Read 2 bytes as a word from 8 bit register and swap the byte order.
- uint16_t readWord16_2c (uint16_t registerAddress)

  Read 2 bytes as a word from 16 bit register and swap the byte order.
- uint32_t readDoubleWord16 (uint16_t registerAddress)

  Read 4 bytes as a double word from 16 bit register.
- bool writeBit8 (uint8_t registerAddress, uint8_t bitNumber, uint8_t data)

  Write a bit to 8 bit register.
- bool writeBit16 (uint16_t registerAddress, uint8_t bitNumber, uint8_t data)

  Write a bit to 16 bit register.
- bool writeBits8 (uint8_t registerAddress, uint8_t bitStart, uint8_t lenght, uint8_t data)

  Write consecutive bits to 8 bit register.
- bool writeBits16 (uint16_t registerAddress, uint8_t bitStart, uint8_t lenght, uint8_t data)

  Write consecutive bits to 16 bit register.
- bool writeByte8 (uint8_t registerAddress, uint8_t data)

  Write a byte to 8 bit register.
- bool writeByte16 (uint16_t registerAddress, uint8_t data)

  Write a byte to 16 bit register.
- bool writeBytes8 (uint8_t registerAddress, uint8_t lenght, uint8_t *data)

  Write consecutive bytes to 8 bit register.
- bool writeBytes16 (uint16_t registerAddress, uint8_t lenght, uint8_t *data)

  Write consecutive bytes to 16 bit register.
- bool writeWord8 (uint8_t registerAddress, uint16_t data)

  Write 2 bytes as a word to 8 bit register.
- bool writeWord16 (uint16_t registerAddress, uint16_t data)

  Write 2 bytes as a word to 16 bit register.
- bool writeDoubleWord (uint16_t registerAddress, uint32_t data)

  Write 4 bytes as a double word to 16 bit register.

Private Attributes

- const uint8_t i2c_bus = 0x1

  I2C bus number.
- const uint8_t i2c_address

  I2C address of the device.
- int fd

  The generic connection to user's chosen I2C hardware.

3.3.1 Detailed Description

Class for controlling the I2C bus.

This class provides an interface for controlling I2C bus using the Raspberry Pi.

Definition at line 31 of file i2c.hpp.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 I2C()

```
I2C::I2C (
            const uint8_t & i2c_bus,
            const uint8_t & i2c_address )
```

Constructor for the QI2C class.

Parameters

i2c_bus	I2C bus number
i2c_address	I2C address of the device

3.3.2.2 ~I2C()

```
I2C::~I2C ( )
```

Destructor for the QI2C class.

3.3.3 Member Function Documentation

3.3.3.1 getFileDescriptor()

```
int I2C::getFileDescriptor ( )  [inline]
```

Gets the file descriptor of the I2C device.

Returns

File descriptor of the I2C device

Definition at line 49 of file i2c.hpp.

```
49 {return fd;}
```

3.3.3.2 i2cClose()

```
void I2C::i2cClose ( )
```

Closes the I2C bus.

3.3.3.3 i2cInit()

```
void I2C::i2cInit ( )
```

Initializes the I2C bus.

3.3.3.4 readBit16()

```
uint8_t I2C::readBit16 (
            uint16_t registerAddress,
            uint8_t bitNumber )
```

Read bit from 16 bit register.

Parameters

registerAddress	The register address
bitNumber	The bit number

Returns

The read bit

3.3.3.5 readBit8()

```
uint8_t I2C::readBit8 (
            uint8_t registerAddress,
            uint8_t bitNumber )
```

Read bit from 8 bit register.

Parameters

registerAddress	The register address
bitNumber	The bit number

Returns

The read bit

3.3.3.6 readBits16()

```
uint8_t I2C::readBits16 (
            uint16_t registerAddress,
            uint8_t bitStart,
            uint8_t length )
```

Read consecutive bits from 16 bit register.

Parameters

registerAddress	The register address
bitStart	The bit start
length	The number of bits to read

Returns

The read bits

3.3.3.7 readBits8()

```
uint8_t I2C::readBits8 (
            uint8_t registerAddress,
            uint8_t bitStart,
            uint8_t length )
```

Read consecutive bits from 8 bit register.

Parameters

registerAddress	The register address
bitStart	The bit start
length	The number of bits to read

Returns

The read bits

3.3.3.8 readByte16()

```
uint8_t I2C::readByte16 (
            uint16_t registerAddress )
```

Read a byte from 16 bit register.

Parameters

registerAddress	The register address

Returns

The read byte

3.3.3.9 readByte8()

```
uint8_t I2C::readByte8 (
            uint8_t registerAddress )
```

Read a byte from 8 bit register.

Parameters

registerAddress	The register address

Returns

The read byte

3.3.3.10 readBytes16()

```
void I2C::readBytes16 (
            uint16_t registerAddress,
            uint8_t length,
            uint8_t * data )
```

Read consecutive bytes from 16 bit register.

Parameters

registerAddress	The register address
length	The number of bytes to read
data	The read bytes

3.3.3.11 readBytes8()

```
void I2C::readBytes8 (
            uint8_t registerAddress,
            uint8_t length,
            uint8_t * data )
```

Read consecutive bytes from 8 bit register.

Parameters

registerAddress	The register address
length	The number of bytes to read
data	The read bytes

3.3.3.12 readDoubleWord16()

```
uint32_t I2C::readDoubleWord16 (
            uint16_t registerAddress )
```

Read 4 bytes as a double word from 16 bit register.

Parameters

registerAddress	The register address

Returns

The read double word

3.3.3.13 readWord16()

```
uint16_t I2C::readWord16 (
            uint16_t registerAddress )
```

Read 2 bytes as a word from 16 bit register.

Parameters

registerAddress	The register address

Returns

The read word

3.3.3.14 readWord16_2c()

```
uint16_t I2C::readWord16_2c (
            uint16_t registerAddress )
```

Read 2 bytes as a word from 16 bit register and swap the byte order.

Parameters

registerAddress	The register address

Returns

The read word

3.3.3.15 readWord8()

```
uint16_t I2C::readWord8 (
            uint8_t registerAddress )
```

Read 2 bytes as a word from 8 bit register.

Parameters

registerAddress	The register address

Returns

The read word

3.3.3.16 readWord8_2c()

```
uint16_t I2C::readWord8_2c (
            uint8_t registerAddress )
```

Read 2 bytes as a word from 8 bit register and swap the byte order.

Parameters

registerAddress	The register address

Returns

The read word

3.3.3.17 writeBit16()

```
bool I2C::writeBit16 (
            uint16_t registerAddress,
            uint8_t bitNumber,
            uint8_t data )
```

Write a bit to 16 bit register.

Parameters

registerAddress	The register address
bitNumber	The bit number
data	The data to write

Returns

True if the operation was successful, false otherwise

3.3.3.18 writeBit8()

```
bool I2C::writeBit8 (
            uint8_t registerAddress,
            uint8_t bitNumber,
            uint8_t data )
```

Write a bit to 8 bit register.

Parameters

registerAddress	The register address
bitNumber	The bit number
data	The data to write

Returns

True if the operation was successful, false otherwise

3.3.3.19 writeBits16()

```
bool I2C::writeBits16 (
            uint16_t registerAddress,
            uint8_t bitStart,
            uint8_t lenght,
            uint8_t data )
```

Write consecutive bits to 16 bit register.

Parameters

registerAddress	The register address
bitStart	The bit start
lenght	The number of bits to write
data	The data to write

Returns

True if the operation was successful, false otherwise

3.3.3.20 writeBits8()

```
bool I2C::writeBits8 (
            uint8_t registerAddress,
            uint8_t bitStart,
            uint8_t lenght,
            uint8_t data )
```

Write consecutive bits to 8 bit register.

Parameters

registerAddress	The register address
bitStart	The bit start
lenght	The number of bits to write
data	The data to write

Returns

True if the operation was successful, false otherwise

3.3.3.21 writeByte16()

```
bool I2C::writeByte16 (
            uint16_t registerAddress,
            uint8_t data )
```

Write a byte to 16 bit register.

Parameters

registerAddress	The register address
data	The data to write

Returns

True if the operation was successful, false otherwise

3.3.3.22 writeByte8()

```
bool I2C::writeByte8 (
            uint8_t registerAddress,
            uint8_t data )
```

Write a byte to 8 bit register.

Parameters

registerAddress	The register address
data	The data to write

Returns

True if the operation was successful, false otherwise

3.3.3.23 writeBytes16()

```
bool I2C::writeBytes16 (
            uint16_t registerAddress,
            uint8_t lenght,
            uint8_t * data )
```

Write consecutive bytes to 16 bit register.

Parameters

registerAddress	The register address
lenght	The number of bytes to write
data	The data to write

Returns

True if the operation was successful, false otherwise

3.3.3.24 writeBytes8()

```
bool I2C::writeBytes8 (
            uint8_t registerAddress,
            uint8_t lenght,
            uint8_t * data )
```

Write consecutive bytes to 8 bit register.

Parameters

registerAddress	The register address
lenght	The number of bytes to write
data	The data to write

Returns

True if the operation was successful, false otherwise

3.3.3.25 writeDoubleWord()

```
bool I2C::writeDoubleWord (
            uint16_t registerAddress,
            uint32_t data )
```

Write 4 bytes as a double word to 16 bit register.

Parameters

registerAddress	The register address
data	The data to write

Returns

True if the operation was successful, false otherwise

3.3.3.26 writeWord16()

```
bool I2C::writeWord16 (
            uint16_t registerAddress,
            uint16_t data )
```

Write 2 bytes as a word to 16 bit register.

Parameters

registerAddress	The register address
data	The data to write

Returns

True if the operation was successful, false otherwise

3.3.3.27 writeWord8()

```
bool I2C::writeWord8 (
            uint8_t registerAddress,
            uint16_t data )
```

Write 2 bytes as a word to 8 bit register.

Parameters

registerAddress	The register address
data	The data to write

Returns

True if the operation was successful, false otherwise

3.3.4 Member Data Documentation

3.3.4.1 fd

```
int I2C::fd  [private]
```

The generic connection to user's chosen I2C hardware.

Definition at line 245 of file i2c.hpp.

3.3.4.2 i2c_address

```
const uint8_t I2C::i2c_address  [private]
```

I2C address of the device.

Definition at line 240 of file i2c.hpp.

3.3.4.3 i2c_bus

```
const uint8_t I2C::i2c_bus = 0x1  [private]
```

I2C bus number.

Definition at line 236 of file i2c.hpp.

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

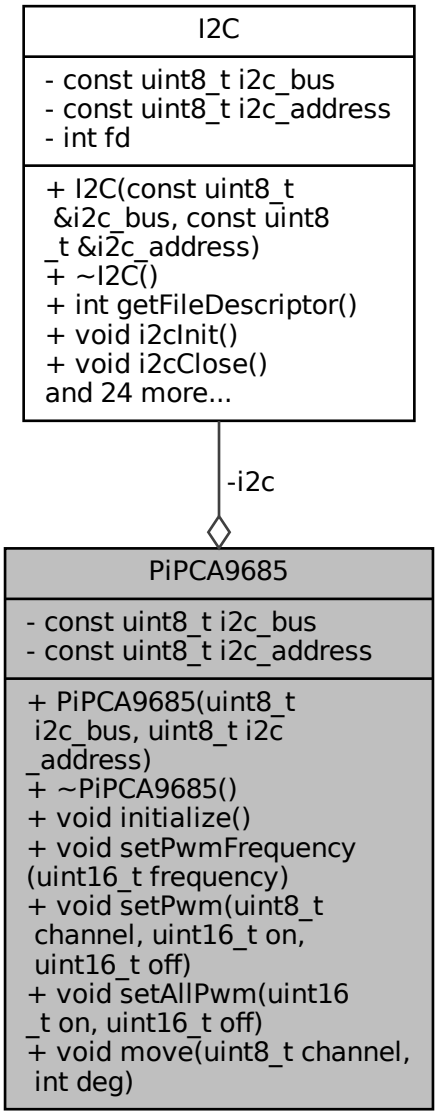
- i2c.hpp

3.4 PiPCA9685 Class Reference

Class for controlling the PCA9685 PWM chip.

```
#include <pipca9685.hpp>
```

Collaboration diagram for PiPCA9685:



Public Member Functions

- PiPCA9685 (uint8_t i2c_bus, uint8_t i2c_address)

  Constructor for the PiPCA9685 class.
- ~PiPCA9685 ()

  Destructor for the PiPCA9685 class.
- void initialize ()

  Initialize the PCA9685 chip.
- void setPwmFrequency (uint16_t frequency)

  Set the PWM frequency for all channels.

- void setPwm (uint8_t channel, uint16_t on, uint16_t off)

  Set the PWM duty cycle for a specific channel.
- void setAllPwm (uint16_t on, uint16_t off)

  Set the PWM duty cycle for all channels.
- void move (uint8_t channel, int deg)

  Move the servo to the specified degree.

Private Attributes

- I2C * i2c

  I2C interface object.
- const uint8_t i2c_bus

  I2C bus number.
- const uint8_t i2c_address

  I2C address of the PCA9685 chip.

3.4.1 Detailed Description

Class for controlling the PCA9685 PWM chip.

This class provides an interface for controlling the PCA9685 PWM chip using the I2C bus on a Raspberry Pi. It allows to change the positions of the servomotors based on the desired angle (in degrees).

Definition at line 92 of file pipca9685.hpp.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 PiPCA9685()

```
PiPCA9685::PiPCA9685 (
            uint8_t i2c_bus,
            uint8_t i2c_address )  [explicit]
```

Constructor for the PiPCA9685 class.

Parameters

i2c_bus	I2C bus number
i2c_address	I2C address of the PCA9685 chip

3.4.2.2 ~PiPCA9685()

```
PiPCA9685::~PiPCA9685 ( )
```

Destructor for the PiPCA9685 class.

3.4.3 Member Function Documentation

3.4.3.1 initialize()

```
void PiPCA9685::initialize ( )
```

Initialize the PCA9685 chip.

3.4.3.2 move()

```
void PiPCA9685::move (
            uint8_t channel,
            int deg )
```

Move the servo to the specified degree.

Parameters

channel	PWM channel (0-15)
deg	Degrees to move

3.4.3.3 setAllPwm()

```
void PiPCA9685::setAllPwm (
            uint16_t on,
            uint16_t off )
```

Set the PWM duty cycle for all channels.

Parameters

on	Duty cycle on time in ticks
off	Duty cycle off time in ticks

3.4.3.4 setPwm()

```
void PiPCA9685::setPwm (
```

```
            uint8_t channel,
            uint16_t on,
            uint16_t off )
```

Set the PWM duty cycle for a specific channel.

Parameters

channel	PWM channel (0-15)
on	Duty cycle on time in ticks
off	Duty cycle off time in ticks

3.4.3.5 setPwmFrequency()

```
void PiPCA9685::setPwmFrequency (
            uint16_t frequency )
```

Set the PWM frequency for all channels.

Parameters

frequency	PWM frequency in Hz

3.4.4 Member Data Documentation

3.4.4.1 i2c

```
I2C* PiPCA9685::i2c  [private]
```

I2C interface object.

Definition at line 141 of file pipca9685.hpp.

3.4.4.2 i2c_address

```
const uint8_t PiPCA9685::i2c_address  [private]
```

I2C address of the PCA9685 chip.

Definition at line 150 of file pipca9685.hpp.

3.4.4.3 i2c_bus

```
const uint8_t PiPCA9685::i2c_bus  [private]
```

I2C bus number.

Definition at line 146 of file pipca9685.hpp.

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

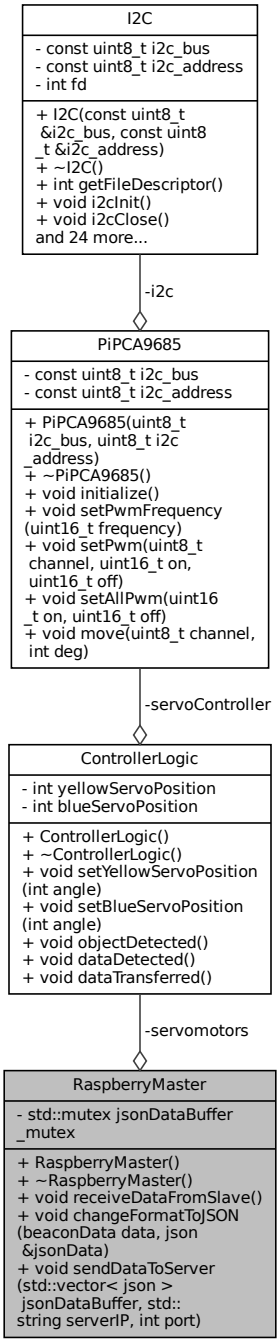
- pipca9685.hpp

3.5 RaspberryMaster Class Reference

The RaspberryMaster class is used for communication between the Master device, the slave device and the TCP remote server. It contains methods for receiving data from the slave device, changing the format of the data to JSON, and sending the data to a server. It also dictates when the servocontroller should perform its actions.

```
#include <raspberrymaster.hpp>
```

Collaboration diagram for RaspberryMaster:



Public Member Functions

- RaspberryMaster ()

  Constructor for the RaspberryMaster class.
- ~RaspberryMaster ()

  Destructor for the RaspberryMaster class.
- void receiveDataFromSlave ()

Method for receiving data from the slave device.

- void changeFormatToJSON (beaconData data, json &jsonData)

  Method for creating a thread for the object detection.
- void sendDataToServer (std::vector< json > jsonDataBuffer, std::string serverIP, int port)

  Method for sending the data to a server.

Private Attributes

- ControllerLogic * servomotors

  Pointer to the ControllerLogic class.
- std::mutex jsonDataBuffer_mutex

  Mutex variable to protect the JSON data buffer in multithreading.

3.5.1 Detailed Description

The RaspberryMaster class is used for communication between the Master device, the slave device and the TCP remote server. It contains methods for receiving data from the slave device, changing the format of the data to JSON, and sending the data to a server. It also dictates when the servocontroller should perform its actions.

Definition at line 58 of file raspberrymaster.hpp.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 RaspberryMaster()

```
RaspberryMaster::RaspberryMaster ( )
```

Constructor for the RaspberryMaster class.

3.5.2.2 ~RaspberryMaster()

```
RaspberryMaster::~RaspberryMaster ( )
```

Destructor for the RaspberryMaster class.

3.5.3 Member Function Documentation

3.5.3.1 changeFormatToJSON()

```
void RaspberryMaster::changeFormatToJSON (
            beaconData data,
            json & jsonData )
```

Method for creating a thread for the object detection.

Parameters

data	The data received from the slave device in the form of a beaconData struct.

3.5.3.2 receiveDataFromSlave()

```
void RaspberryMaster::receiveDataFromSlave ( )
```

Method for receiving data from the slave device.

3.5.3.3 sendDataToServer()

```
void RaspberryMaster::sendDataToServer (
            std::vector< json > jsonDataBuffer,
            std::string serverIP,
            int port )
```

Method for sending the data to a server.

Parameters

jsonData	The data in JSON format.
serverIP	The IP address of the server.
port	The port number that the data will be sent to.

3.5.4 Member Data Documentation

3.5.4.1 jsonDataBuffer_mutex

```
std::mutex RaspberryMaster::jsonDataBuffer_mutex  [private]
```

Mutex variable to protect the JSON data buffer in multithreading.

Definition at line 98 of file raspberrymaster.hpp.

3.5.4.2 servomotors

```
ControllerLogic* RaspberryMaster::servomotors  [private]
```

Pointer to the ControllerLogic class.

Definition at line 94 of file raspberrymaster.hpp.

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

- raspberrymaster.hpp

Chapter 4

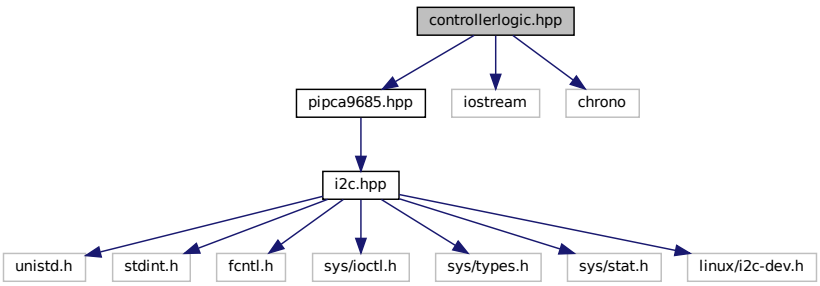
File Documentation

4.1 controllerlogic.hpp File Reference

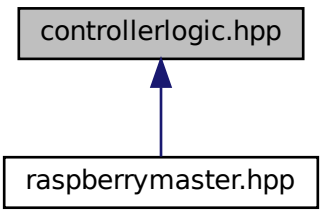
This file contains the class definition for ControllerLogic.

```
#include "pipca9685.hpp"
#include <iostream>
#include <chrono>
```

Include dependency graph for controllerlogic.hpp:



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



Classes

- class ControllerLogic

  Class for describing the actions that are to be performed by the PCA9685 controller.

4.1.1 Detailed Description

This file contains the class definition for ControllerLogic.

Author

Georges Schuhl

Nicolas Chataignon

Houssein Mariam

Version

1.0

Date

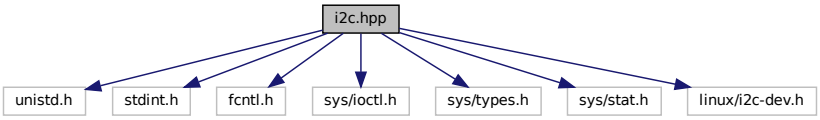
17 january 2023

4.2 i2c.hpp File Reference

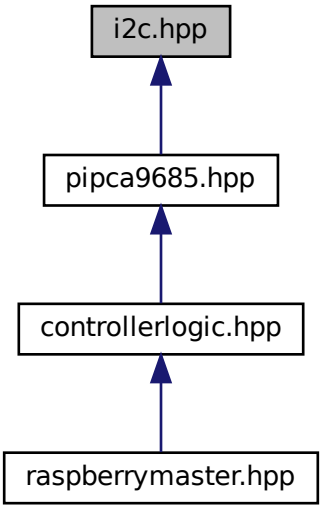
This file contains the class definition for I2C.

```
#include <unistd.h>
#include <stdint.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <linux/i2c-dev.h>
```

Include dependency graph for i2c.hpp:



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



Classes

- class I2C

  Class for controlling the I2C bus.

4.2.1 Detailed Description

This file contains the class definition for I2C.

Author

Georges Schuhl
Nicolas Chataignon
Houssein Mariam
manfredipist https://github.com/manfredipist/QI2CProtocol

Version

1.0

Date

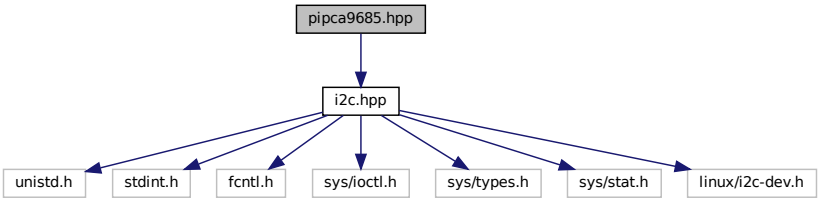
17 january 2023

4.3 pipca9685.hpp File Reference

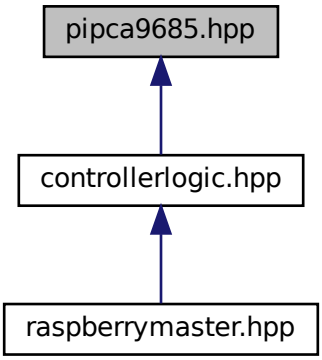
This file contains the class definition for PiPCA9685.

```
#include "i2c.hpp"
```

Include dependency graph for pipca9685.hpp:



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



Classes

- class PiPCA9685

  Class for controlling the PCA9685 PWM chip.

Macros

- #define MODE1 0x00

  Register address for Mode 1.
- #define MODE2 0x01

  Register address for Mode 2.
- #define SUBADR1 0x02

Register address for Subaddress 1.

- #define SUBADR2 0x03

  Register address for Subaddress 2.
- #define SUBADR3 0x04

  Register address for Subaddress 3.
- #define PRESCALE 0xFE

  Register address for Prescale.
- #define LED0_ON_L 0x06

  Register address for LED 0 on low byte.
- #define LED0_ON_H 0x07

  Register address for LED 0 on high byte.
- #define LED0_OFF_L 0x08

  Register address for LED 0 off low byte.
- #define LED0_OFF_H 0x09

  Register address for LED 0 off high byte.
- #define ALL_LED_ON_L 0xFA

  Register address for all LED on low byte.
- #define ALL_LED_ON_H 0xFB

  Register address for all LED on high byte.
- #define ALL_LED_OFF_L 0xFC

  Register address for all LED off low byte.
- #define ALL_LED_OFF_H 0xFD

  Register address for all LED off high byte.
- #define RESTART 0x80

  Bit for restarting.
- #define SLEEP 0x10

  Bit for sleeping.
- #define ALLCALL 0x01

  Bit for all call.
- #define INVRT 0x10

  Bit for inverting.
- #define OUTDRV 0x04

  Bit for output driving.
- #define PAN 0

  Channel for pan control.
- #define TILT 1

  Channel for tilt control.
- #define FREQUENCY 50

  PWM frequency.
- #define CLOCKFREQ 25000000

  Clock frequency.
- #define PANOFFSET 1

  Pan offset.
- #define PANSCALE 1.4

  Pan scale.
- #define TILTOFFSET 30

  Tilt offset.
- #define TILTSCALE 1.43

  Tilt scale.
- #define PANMAX 270

  Maximum pan angle.

- #define PANMIN 90
  Minimum pan angle.
- #define TILTMAX 90
  Maximum tilt angle.
- #define TILTMIN -45
  Minimum tilt angle.

4.3.1 Detailed Description

This file contains the class definition for PiPCA9685.

Author

Georges Schuhl
Nicolas Chataignon
Houssein Mariam
manfredipist https://github.com/manfredipist/QI2CProtocol

Version

1.0

Date

17 january 2023

4.3.2 Macro Definition Documentation

4.3.2.1 ALL_LED_OFF_H

```
#define ALL_LED_OFF_H 0xFD
```

Register address for all LED off high byte.

Definition at line 45 of file pipca9685.hpp.

4.3.2.2 ALL_LED_OFF_L

```
#define ALL_LED_OFF_L 0xFC
```

Register address for all LED off low byte.

Definition at line 43 of file pipca9685.hpp.

4.3.2.3 ALL_LED_ON_H

```
#define ALL_LED_ON_H 0xFB
```

Register address for all LED on high byte.

Definition at line 41 of file pipca9685.hpp.

4.3.2.4 ALL_LED_ON_L

```
#define ALL_LED_ON_L 0xFA
```

Register address for all LED on low byte.

Definition at line 39 of file pipca9685.hpp.

4.3.2.5 ALLCALL

```
#define ALLCALL 0x01
```

Bit for all call.

Definition at line 51 of file pipca9685.hpp.

4.3.2.6 CLOCKFREQ

```
#define CLOCKFREQ 25000000
```

Clock frequency.

Definition at line 64 of file pipca9685.hpp.

4.3.2.7 FREQUENCY

```
#define FREQUENCY 50
```

PWM frequency.

Definition at line 62 of file pipca9685.hpp.

4.3.2.8 INVRT

```
#define INVRT 0x10
```

Bit for inverting.

Definition at line 53 of file pipca9685.hpp.

4.3.2.9 LED0_OFF_H

```
#define LED0_OFF_H 0x09
```

Register address for LED 0 off high byte.

Definition at line 37 of file pipca9685.hpp.

4.3.2.10 LED0_OFF_L

```
#define LED0_OFF_L 0x08
```

Register address for LED 0 off low byte.

Definition at line 35 of file pipca9685.hpp.

4.3.2.11 LED0_ON_H

```
#define LED0_ON_H 0x07
```

Register address for LED 0 on high byte.

Definition at line 33 of file pipca9685.hpp.

4.3.2.12 LED0_ON_L

```
#define LED0_ON_L 0x06
```

Register address for LED 0 on low byte.

Definition at line 31 of file pipca9685.hpp.

4.3.2.13 MODE1

```
#define MODE1 0x00
```

Register address for Mode 1.

Definition at line 19 of file pipca9685.hpp.

4.3.2.14 MODE2

```
#define MODE2 0x01
```

Register address for Mode 2.

Definition at line 21 of file pipca9685.hpp.

4.3.2.15 OUTDRV

```
#define OUTDRV 0x04
```

Bit for output driving.

Definition at line 55 of file pipca9685.hpp.

4.3.2.16 PAN

```
#define PAN 0
```

Channel for pan control.

Definition at line 58 of file pipca9685.hpp.

4.3.2.17 PANMAX

```
#define PANMAX 270
```

Maximum pan angle.

Definition at line 74 of file pipca9685.hpp.

4.3.2.18 PANMIN

```
#define PANMIN 90
```

Minimum pan angle.

Definition at line 76 of file pipca9685.hpp.

4.3.2.19 PANOFFSET

```
#define PANOFFSET 1
```

Pan offset.

Definition at line 66 of file pipca9685.hpp.

4.3.2.20 PANSCALE

```
#define PANSCALE 1.4
```

Pan scale.

Definition at line 68 of file pipca9685.hpp.

4.3.2.21 PRESCALE

```
#define PRESCALE 0xFE
```

Register address for Prescale.

Definition at line 29 of file pipca9685.hpp.

4.3.2.22 RESTART

```
#define RESTART 0x80
```

Bit for restarting.

Definition at line 47 of file pipca9685.hpp.

4.3.2.23 SLEEP

```
#define SLEEP 0x10
```

Bit for sleeping.

Definition at line 49 of file pipca9685.hpp.

4.3.2.24 SUBADR1

```
#define SUBADR1 0x02
```

Register address for Subaddress 1.

Definition at line 23 of file pipca9685.hpp.

4.3.2.25 SUBADR2

```
#define SUBADR2 0x03
```

Register address for Subaddress 2.

Definition at line 25 of file pipca9685.hpp.

4.3.2.26 SUBADR3

```
#define SUBADR3 0x04
```

Register address for Subaddress 3.

Definition at line 27 of file pipca9685.hpp.

4.3.2.27 TILT

```
#define TILT 1
```

Channel for tilt control.

Definition at line 60 of file pipca9685.hpp.

4.3.2.28 TILTMAX

```
#define TILTMAX 90
```

Maximum tilt angle.

Definition at line 78 of file pipca9685.hpp.

4.3.2.29 TILTMIN

```
#define TILTMIN -45
```

Minimum tilt angle.

Definition at line 80 of file pipca9685.hpp.

4.3.2.30 TILTOFFSET

```
#define TILTOFFSET 30
```

Tilt offset.

Definition at line 70 of file pipca9685.hpp.

4.3.2.31 TILTSCALE

```
#define TILTSCALE 1.43
```

Tilt scale.

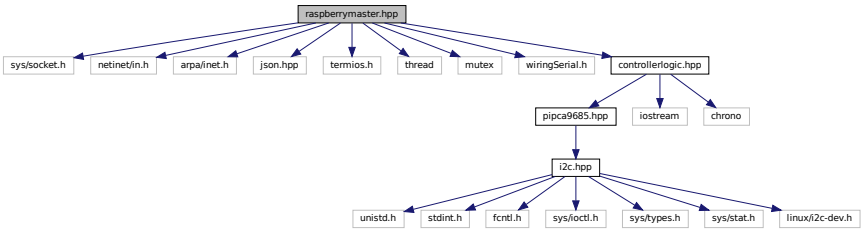
Definition at line 72 of file pipca9685.hpp.

4.4 raspberrymaster.hpp File Reference

This file contains the declaration of the RaspberryMaster class and the beaconData struct. The RaspberryMaster class is used for communication between the Raspberry Pi and the slave device. The beaconData struct contains data received from the slave device, such as temperature, pressure, humidity, etc.

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <json.hpp>
#include <termios.h>
#include <thread>
#include <mutex>
#include "wiringSerial.h"
#include "controllerlogic.hpp"
```

Include dependency graph for raspberrymaster.hpp:



Classes

- struct beaconData

  The beaconData struct contains data received from the slave device, such as temperature, pressure, humidity, x, y, z axes positions, detection status and a timestamp.
- class RaspberryMaster

  The RaspberryMaster class is used for communication between the Master device, the slave device and the TCP remote server. It contains methods for receiving data from the slave device, changing the format of the data to JSON, and sending the data to a server. It also dictates when the servocontroller should perform its actions.

Typedefs

- using json = nlohmann::json

4.4.1 Detailed Description

This file contains the declaration of the RaspberryMaster class and the beaconData struct. The RaspberryMaster class is used for communication between the Raspberry Pi and the slave device. The beaconData struct contains data received from the slave device, such as temperature, pressure, humidity, etc.

Author

Georges Schuhl

Nicolas Chataignon

Houssein Mariam

Version

1.0

Date

17 january 2023

4.4.2 Typedef Documentation

4.4.2.1 json

```
using json = nlohmann::json
```

Definition at line 27 of file raspberrymaster.hpp.

Index