B1 bazen Laser Tag

1

Generated by Doxygen 1.8.11

Contents

Chapter 1

Laser tag

This is doxygen documentation for our school project named THEMA DEVICES.

THEMA DEVICES is a group project about building a laser tag game with RTOS. This is the proof of concept documentation.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

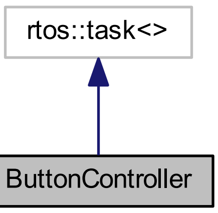
Class Documentation

5.1 ButtonController Class Reference

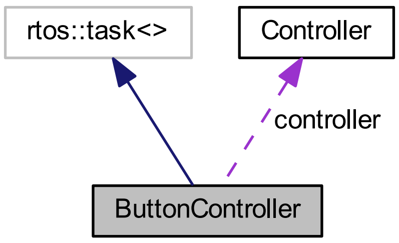
can be interpreted as playing state and will handle each event during his state.

```
#include <buttonController.h>
```

Inheritance diagram for ButtonController:



Collaboration diagram for ButtonController:



Public Member Functions

- ButtonController (Controller ∗ctrl, hwlib::pin_out &gnd, hwlib::pin_out &vlt, hwlib::pin_in &b)
- void set_listener (Controller ∗ctrl)

Private Member Functions

- void main ()

Private Attributes

- Controller ∗ controller
- hwlib::pin_out & ground
- hwlib::pin_out & voltage
- hwlib::pin_in & button
- rtos::clock clock

5.1.1 Detailed Description

can be interpreted as playing state and will handle each event during his state.

ButtonController Task. Can set flags with interfaces form referenced controllers.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 ButtonController::ButtonController (**Controller** ∗ *ctrl,* hwlib::pin_out & *gnd,* hwlib::pin_out & *vlt,* hwlib::pin_in & *b*)

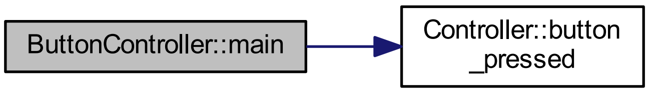
Constructor ButtonController. /param ctrl reference for using controller interface /param gnd reference pin requires output pin /param vlt reference pin requires output pin /param b reference pin requires input pin

5.1.3 Member Function Documentation

5.1.3.1 void ButtonController::main ( ) `[private]`

RTOS task function

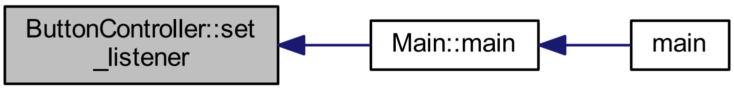
Here is the call graph for this function:



5.1.3.2 void ButtonController::set_listener (Controller ∗ *ctrl*) [inline]

Sets the controller the button will send messages to

Here is the caller graph for this function:



5.1.4 Member Data Documentation

5.1.4.1 hwlib::pin_in& ButtonController::button [private]

5.1.4.2 rtos::clock ButtonController::clock [private]

clock for polling button press

5.1.4.3 Controller∗ ButtonController::controller [private]

abstract controller to use interface function

5.1.4.4 hwlib::pin_out& ButtonController::ground [private]

pin settings for the button

5.1.4.5 hwlib::pin_out& ButtonController::voltage [private]

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

- src/tasks/buttonController.h
- src/tasks/buttonController.cpp

5.2 Command Class Reference

Command that handles decoding and encoding of the IR commands This class can be instantiated with a short If this happens it will automatically decode the short into readable data like the sender and the actual data.

`#include <command.h>`

Public Member Functions

- void print_command ()
- short encode ()

    Takes the sender and data and turns it into a short encoded with the data Adds a startbit to a 0 initialized short.
- bool get_error ()

    returns error value
- int get_sender ()

    returns the sender value
- void set_sender (int sender)

    sets sender value
- int get_data ()

    returns the data value
- void set_data (int data)

    sets the data value
- Command ()

    empty constructor for Command Empty constructor that initializes sender and data with -1
- Command (short bits)

    Constructor with short decode the bits param into logical sender and data values.
- Command (int sender, int data)

    Constructor that takes a data and a sender.

Private Member Functions

- void decode (short bits)

    decodes short into sender and data Takes the short bits and reads them like binary in MSB The protocol is defined at `https://cursussen.sharepoint.hu.nl/fnt/36/TCTI-V2THDE-16/Studiemateriaal/V2↩ THDE%20-%20Casus%20lasertag%202016-2017.pdf` *PAGE 4*
- bool valid_checksum (short bits)

    Validates the checksum of the given bits The last 5 bits become the checksum The checksum is valid when the XOR of the first till last bit from the id and the first til last bit from the data are equal to the first bit of the checksum.

Private Attributes

- int sender

    the sender of the command
- int data

    the actual data part of the command
- bool error = false

    If the command decoded successfully.

5.2.1 Detailed Description

Command that handles decoding and encoding of the IR commands This class can be instantiated with a short If this happens it will automatically decode the short into readable data like the sender and the actual data.

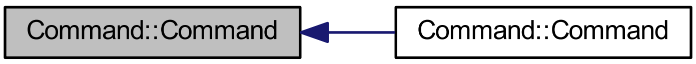
Command It will also generate en validate checksums for the commands

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Command::Command ( )

empty constructor for Command Empty constructor that initializes sender and data with -1

Here is the caller graph for this function:



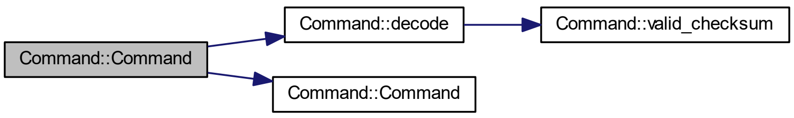
5.2.2.2 Command::Command (short *bits*) [inline]

Constructor with short decode the bits param into logical sender and data values.

Parameters

bits	The short that will be decoded into usable data

Here is the call graph for this function:



5.2.2.3 Command::Command (int *sender,* int *data*)

Constructor that takes a data and a sender.

Parameters

sender	the sender part of the command
data	the data part of the command

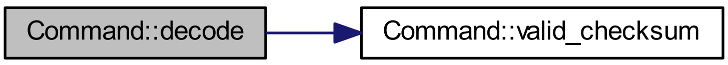
5.2.3 Member Function Documentation

5.2.3.1 void Command::decode (short *bits*) [private]

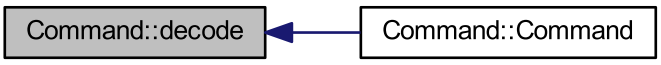
decodes short into sender and data Takes the short bits and reads them like binary in MSB The protocol is defined at https://cursussen.sharepoint.hu.nl/fnt/36/TCTI-V2THDE-16/Studiemateriaal/↩
V2THDE%20-%20Casus%20lasertag%202016-2017.pdf PAGE 4

from left to right the first bit is the start bit Then the next 5 bits is the actual sender ID After that the next 5 bits is the actual data part. The remaining bits become the checksum

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.3.2 short Command::encode ( )

Takes the sender and data and turns it into a short encoded with the data Adds a startbit to a 0 initialized short.

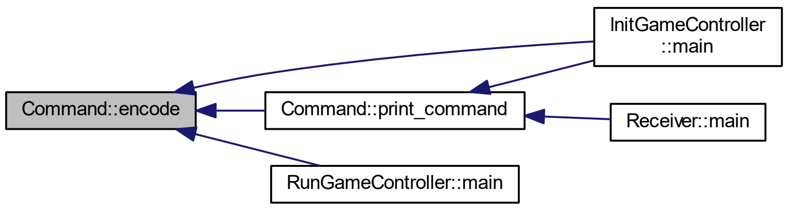
Decodes the int from sender into a binary value and adds it next to the short startbit

Decodes the int from the data into binary value and add it next to the short sender bits

Generates a checksum based on the binary XOR of sender and data and add it nex to the short data bits

Checksum generation sources https://cursussen.sharepoint.hu.nl/fnt/36/TCTI-V2THD↩
E-16/Studiemateriaal/V2THDE%20-%20Casus%20lasertag%202016-2017.pdf

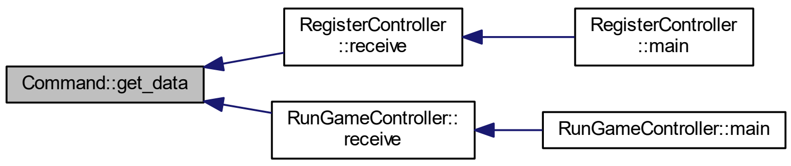
Here is the caller graph for this function:



5.2.3.3 int Command::get_data ( )

returns the data value

Here is the caller graph for this function:



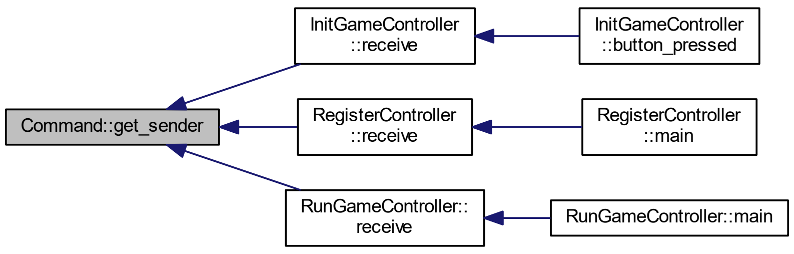
5.2.3.4 bool Command::get_error ( )

returns error value

5.2.3.5 int Command::get_sender ( )

returns the sender value

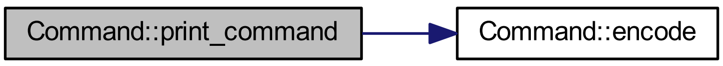
Here is the caller graph for this function:



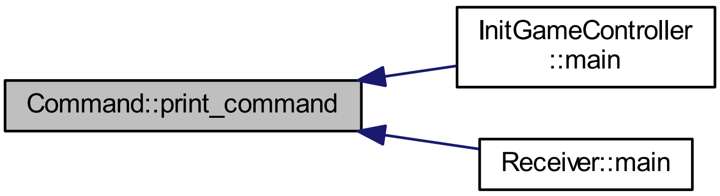
5.2.3.6 void Command::print_command ( )

/brief helper that prints out a decoded version and encoded version of the command

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.3.7 void Command::set_data (int *data*)

sets the data value

5.2.3.8 void Command::set_sender (int *sender*)

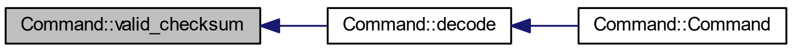
sets sender value

5.2.3.9 bool Command::valid_checksum (short *bits*) [private]

Validates the checksum of the given bits The last 5 bits become the checksum The checksum is valid when the XOR of the first till last bit from the id and the first til last bit from the data are equal to the first bit of the checksum.

Example ID DATA CHECKSUM 1 00100 10100 10000 00100 10100 XOR 10000

Here is the caller graph for this function:



5.2.4 Member Data Documentation

5.2.4.1 int Command::data `[private]`

the actual data part of the command

5.2.4.2 bool Command::error = false `[private]`

If the command decoded successfully.

5.2.4.3 int Command::sender `[private]`

the sender of the command

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

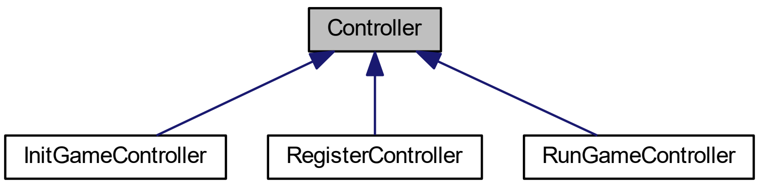
- src/tasks/command.h
- src/tasks/command.cpp

5.3 Controller Class Reference

will be implented by each state

```
#include <controller.h>
```

Inheritance diagram for Controller:



Public Member Functions

- Controller ()
- virtual void receive (Command c)=0
- virtual void enable ()=0
- virtual void button_pressed ()=0
- virtual const char ∗ get_name ()=0

5.3.1 Detailed Description

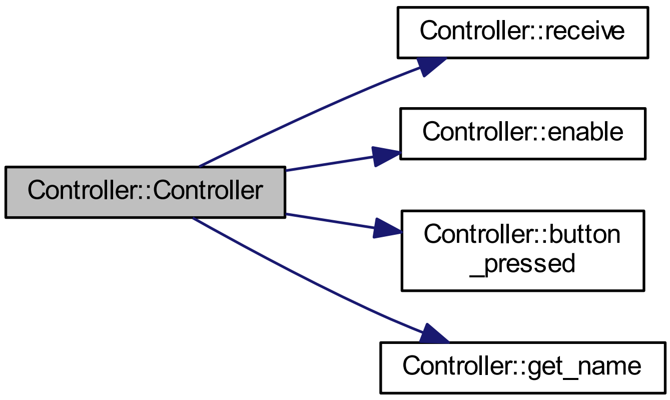
will be implented by each state

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Controller::Controller ( ) [inline]

Controller Constructor

Here is the call graph for this function:



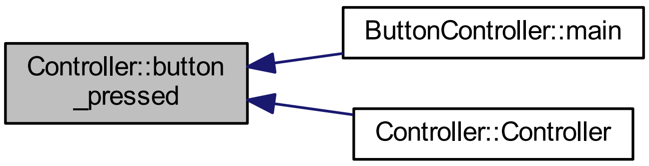
5.3.3 Member Function Documentation

5.3.3.1 virtual void Controller::button_pressed ( ) [pure virtual]

virtual function to use button_pressed function task

Implemented in RunGameController, InitGameController, and RegisterController.

Here is the caller graph for this function:

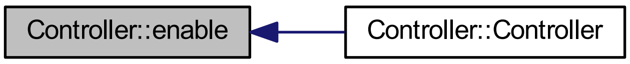


5.3.3.2 virtual void Controller::enable ( ) [pure virtual]

virtual function to enable the class function task

Implemented in InitGameController, RunGameController, and RegisterController.

Here is the caller graph for this function:

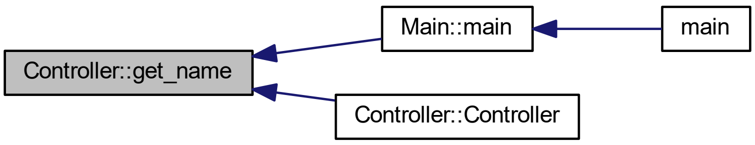


5.3.3.3 virtual const char∗ Controller::get_name ( ) [pure virtual]

virtual function for getting class name

Implemented in InitGameController, RunGameController, and RegisterController.

Here is the caller graph for this function:

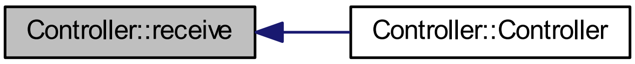


5.3.3.4 virtual void Controller::receive (Command *c*) [pure virtual]

virtual function for received data from the receiver

Implemented in InitGameController, RunGameController, and RegisterController.

Here is the caller graph for this function:



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

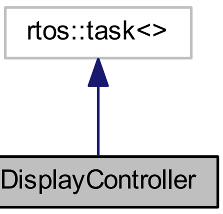
- src/tasks/controller.h

5.4 DisplayController Class Reference

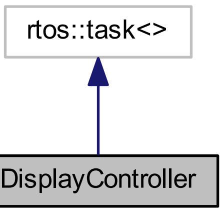
will be used as communcation controller between oled boundary and other game state controllers

`#include <displayController.h>`

Inheritance diagram for DisplayController:



Collaboration diagram for DisplayController:



Public Member Functions

- DisplayController (hwlib::glcd_oled_buffered &o)
- void displayText (const char $*$)

Private Member Functions

- void main ()

Private Attributes

- hwlib::glcd_oled_buffered & oled
- rtos::channel< char, 2048 > buffer
- rtos::timer timer_screen
- rtos::flag clearFlag
- rtos::flag flushFlag

5.4.1 Detailed Description

will be used as communcation controller between oled boundary and other game state controllers

5.4.2 Constructor & Destructor Documentation

5.4.2.1 DisplayController::DisplayController (hwlib::glcd_oled_buffered & *o*)

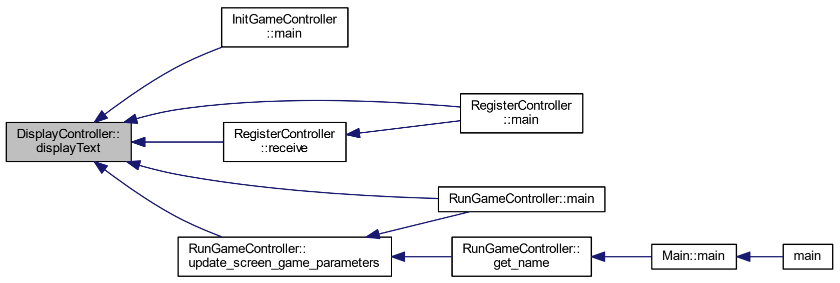
DisplayController Constructor /param hwlib::glc_oled_buffered &c will be the oled boundary object for writing on screen

5.4.3 Member Function Documentation

5.4.3.1 void DisplayController::displayText (const char $*$ *txt*)

interface function to write string on screen

Here is the caller graph for this function:


InitGameController ::main
DisplayController:: displayText
RegisterController ::receive
RegisterController ::main
RunGameController::main
RunGameController:: update_screen_game_parameters
RunGameController:: get_name
Main::main
main

5.4.3.2 void DisplayController::main ( ) [private]

RTOS task function

5.4.4 Member Data Documentation

5.4.4.1 rtos::channel< char, 2048 > DisplayController::buffer [private]

RTOS buffer for storing the char to write to oled display

5.4.4.2 rtos::flag DisplayController::clearFlag [private]

RTOS flag to clear screen

5.4.4.3 rtos::flag DisplayController::flushFlag [private]

RTOS flag to flush screen

5.4.4.4 hwlib::glcd_oled_buffered& DisplayController::oled [private]

oled boundary reference

5.4.4.5 rtos::timer DisplayController::timer_screen [private]

RTOS timer for updating screen

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

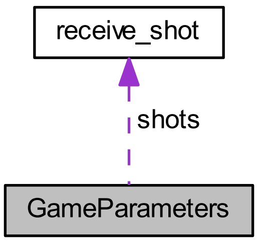
- src/tasks/displayController.h
- src/tasks/displayController.cpp

5.5 GameParameters Class Reference

GameParameters entity object will contain the player data.

```
#include <gameParameters.h>
```

Collaboration diagram for GameParameters:



Public Member Functions

- GameParameters ()
- void add_received_shot (int player_id, int weapon_id)

Public Attributes

- int id
- int health = 100
- int weapon
- int game_time

Private Attributes

- int shots_taken
- receive_shot shots [20]

5.5.1 Detailed Description

GameParameters entity object will contain the player data.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 GameParameters::GameParameters ( )

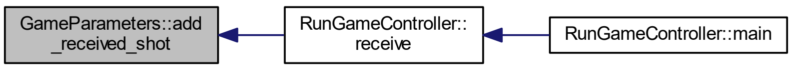
GameParameter Constructor

5.5.3 Member Function Documentation

5.5.3.1 void GameParameters::add_received_shot (int *player_id,* int *weapon_id*)

function to store the received shot data /param player_id int for player id /param weapon_id int for weapon id

Here is the caller graph for this function:


GameParameters::add _received_shot
RunGameController:: receive
RunGameController::main

5.5.4 Member Data Documentation

5.5.4.1 int GameParameters::game_time

game time

5.5.4.2 int GameParameters::health = 100

int health

5.5.4.3 int GameParameters::id

int player id

5.5.4.4 receive_shot GameParameters::shots[20] [private]

received shots array

5.5.4.5 int GameParameters::shots_taken [private]

amount of shot taken

5.5.4.6 int GameParameters::weapon

weapon id

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

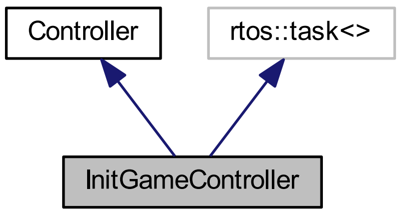
- src/entities/gameParameters.h
- src/entities/gameParameters.cpp

5.6 InitGameController Class Reference

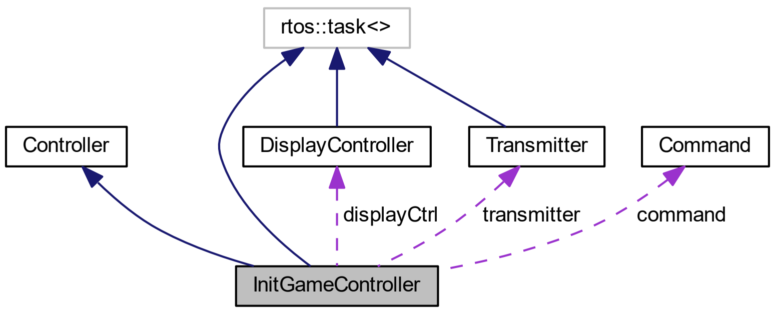
can be interpretted as playing state and will handle each event during his state.

`#include <initGameController.h>`

Inheritance diagram for InitGameController:



Collaboration diagram for InitGameController:



Public Member Functions

- void enable ()
- void button_pressed ()
- void receive (Command c)
- const char $*$ get_name ()
- InitGameController (Transmitter &transmitter, hwlib::keypad$<$ 16 $>$ &keypad, DisplayController &displayCtrl)

Private Member Functions

- int valid_id (char first, char second)
- void main ()

Private Attributes

- Transmitter & transmitter
- hwlib::keypad$<$ 16 $>$ & keypad
- DisplayController & displayCtrl
- rtos::flag enabled
- rtos::flag command_available
- Command command
- int player_id
- int weapon_id
- short custom_command = 0
- char command_full

5.6.1 Detailed Description

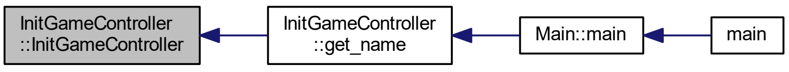
can be interpretted as playing state and will handle each event during his state.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 InitGameController::InitGameController (**Transmitter** & *transmitter,* hwlib::keypad< 16 > & *keypad,* **DisplayController** & *displayCtrl*)

InitGameController constructor /param Transmitter &transmitter is a boundary object to send data /param hwlib↩ ::keypad<16> &keypad is HWLIB boundary for keypad input /param DisplayController &displayCtrl reference to display controller for handling text

Here is the caller graph for this function:


InitGameController ::InitGameController
InitGameController ::get_name
Main::main
main

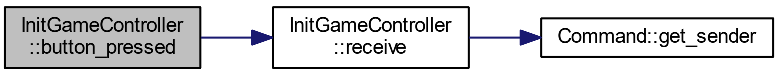
5.6.3 Member Function Documentation

5.6.3.1 void InitGameController::button_pressed ( ) [inline], [virtual]

interface function that's not used

Implements Controller.

Here is the call graph for this function:

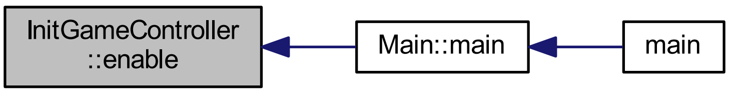

InitGameController ::button_pressed
InitGameController ::receive
Command::get_sender

5.6.3.2 void InitGameController::enable ( ) [virtual]

interface function to set flag that will activate task

Implements Controller.

Here is the caller graph for this function:

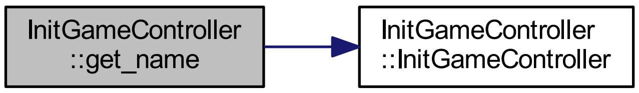


5.6.3.3 const char∗ InitGameController::get_name ( ) [inline], [virtual]

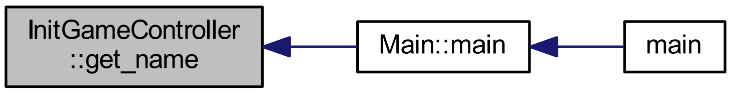
interface function for getting controller name

Implements Controller.

Here is the call graph for this function:


InitGameController ::get_name
InitGameController ::InitGameController

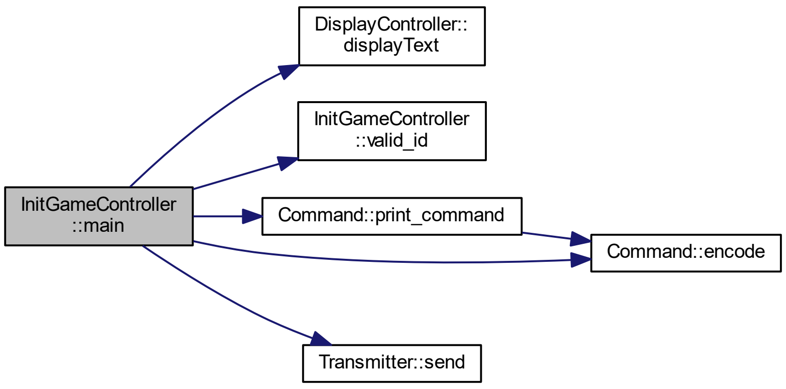
Here is the caller graph for this function:



5.6.3.4 void InitGameController::main ( ) [private]

RTOS main task function

Here is the call graph for this function:

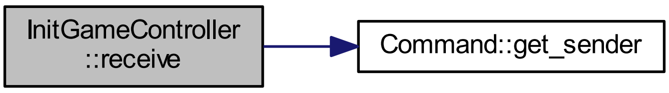

DisplayController:: displayText
InitGameController ::valid_id
InitGameController ::main
Command::print_command
Command::encode
Transmitter::send

5.6.3.5 void InitGameController::receive (**Command *c***) `[virtual]`

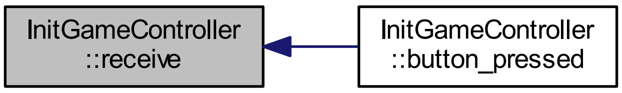
interface function for receiving command /param Command c

Implements Controller.

Here is the call graph for this function:



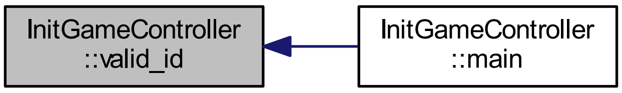
Here is the caller graph for this function:


InitGameController ::receive
InitGameController ::button_pressed

5.6.3.6 int InitGameController::valid_id (char *first*, char *second*) `[private]`

function valid_id will combine 2 input chars from keypad to 1 integer value. integer value will be validated. Must be between 1 and 31. When true, return the valid id. On failure, return 0.

Here is the caller graph for this function:



5.6.4 Member Data Documentation

5.6.4.1 Command InitGameController::command [private]

COMMAND class for encoding and decoding bits

5.6.4.2 rtos::flag InitGameController::command_available [private]

RTOS flag foor command available

5.6.4.3 char InitGameController::command_full [private]

command_full present a boolean value. Used to check if player

5.6.4.4 short InitGameController::custom_command = 0 [private]

short custom_command will be the inputted command by leader with keypad.

5.6.4.5 DisplayController& InitGameController::displayCtrl [private]

displayController to display text on oled

5.6.4.6 rtos::flag InitGameController::enabled [private]

RTOS enable flag to atctivate task

5.6.4.7 hwlib::keypad<16>& InitGameController::keypad [private]

keypad boundary for getting input

5.6.4.8 int InitGameController::player_id [private]

int player_id is NOT admin id! it will be the player id to give

5.6.4.9 Transmitter& InitGameController::transmitter [private]

transmitter boundary for sending data

5.6.4.10 int InitGameController::weapon_id [private]

int weapon_id is weapon id to give

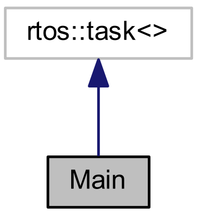
The documentation for this class was generated from the following files:

- src/tasks/initGameController.h
- src/tasks/initGameController.cpp

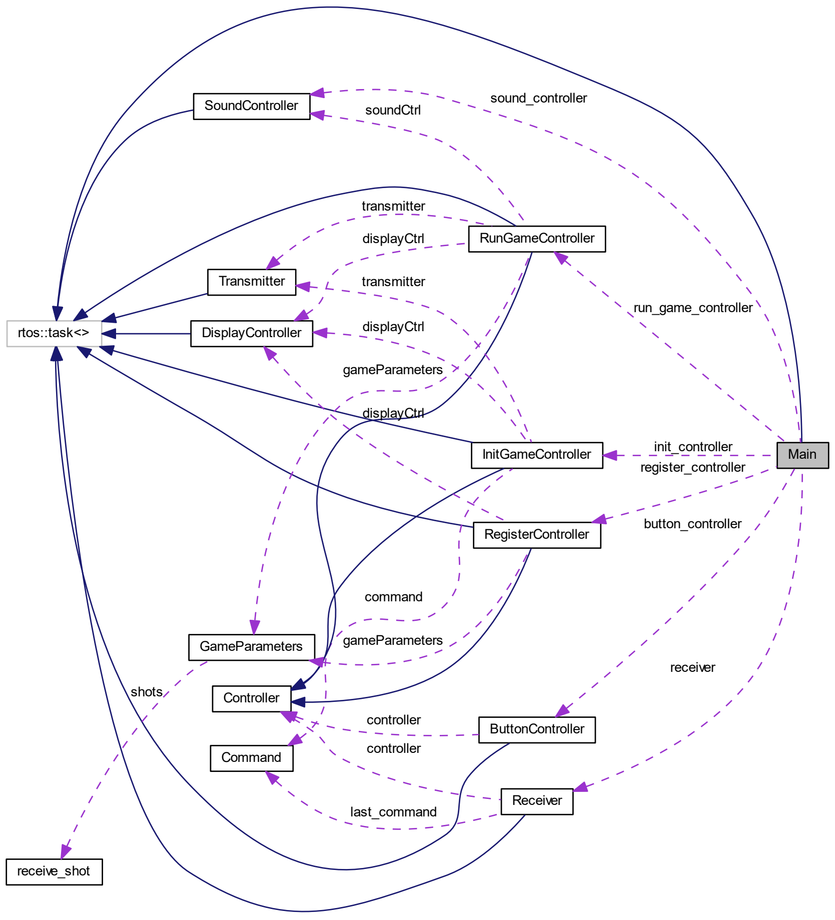
5.7 Main Class Reference

This class will resume and suspend running tasks based on it's current state.

Inheritance diagram for Main:



Collaboration diagram for Main:



Public Member Functions

- Main (Receiver &r, ButtonController &b, InitGameController &i, RegisterController &reg, RunGameController &run, SoundController &sound)

  Constructor for the Main class.

Private Member Functions

- void main ()

Private Attributes

- Receiver & receiver
- ButtonController & button_controller
- InitGameController & init_controller
- RegisterController & register_controller
- RunGameController & run_game_controller
- SoundController & sound_controller

5.7.1 Detailed Description

This class will resume and suspend running tasks based on it's current state.

5.7.2 Constructor & Destructor Documentation

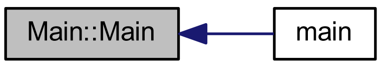
5.7.2.1 Main::Main (**Receiver** & *r*, **ButtonController** & *b*, **InitGameController** & *i*, **RegisterController** & *reg*, **RunGameController** & *run*, **SoundController** & *sound*) `[inline]`

Constructor for the Main class.

Parameters

r	a Reference to a existing instance of a Receiver task
i	a Reference to a existing instance of a InitGameController task
reg	a Reference to a existing instance of a RegisterController task
run	a Reference to a existing instance of a RunGameFController task

Here is the caller graph for this function:

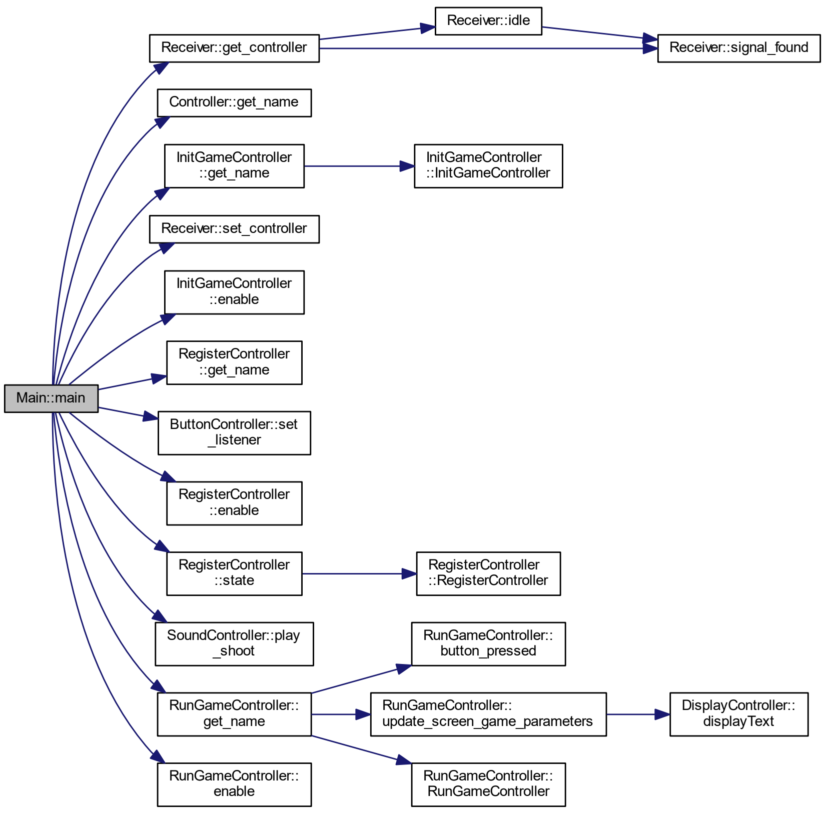


5.7.3 Member Function Documentation

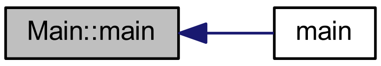
5.7.3.1 void Main::main ( ) `[inline], [private]`

Main loop that the rtos tasks runs and suspends It enables and disables running tasks based on the current state. It will also swap the listeners for the receiver based on the current state.

Here is the call graph for this function:


Main::main
Receiver::get_controller
Receiver::idle
Receiver::signal_found
Controller::get_name
InitGameController ::get_name
InitGameController ::InitGameController
Receiver::set_controller
InitGameController ::enable
RegisterController ::get_name
ButtonController::set _listener
RegisterController ::enable
RegisterController ::state
RegisterController ::RegisterController
SoundController::play _shoot
RunGameController:: button_pressed
RunGameController:: get_name
RunGameController:: update_screen_game_parameters
DisplayController:: displayText
RunGameController:: enable
RunGameController:: RunGameController

Here is the caller graph for this function:



5.7.4 Member Data Documentation

5.7.4.1 ButtonController& Main::button_controller [private]

Reference to a ButtonController task

5.7.4.2 InitGameController& Main::init_controller [private]

Reference to a InitController task

5.7.4.3 Receiver& Main::receiver [private]

Reference to a Receiver task

5.7.4.4 RegisterController& Main::register_controller [private]

Reference to a RegisterController task

5.7.4.5 RunGameController& Main::run_game_controller [private]

Reference to a RunGameController task

5.7.4.6 SoundController& Main::sound_controller [private]

Reference to a SoundController task;

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

- src/main.cpp

5.8 receive_shot Struct Reference

receive_shot structure. Will be used for class GameParameters. Each structure contains the player id and weapon id from the received hit

```
#include <gameParameters.h>
```

Public Attributes

- int player_id
- int weapon_id

5.8.1 Detailed Description

receive_shot structure. Will be used for class GameParameters. Each structure contains the player id and weapon id from the received hit

5.8.2 Member Data Documentation

5.8.2.1 int receive_shot::player_id

play id from the received hit

5.8.2.2 int receive_shot::weapon_id

weapon id from the received hit

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

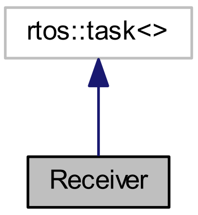
- src/entities/gameParameters.h

5.9 Receiver Class Reference

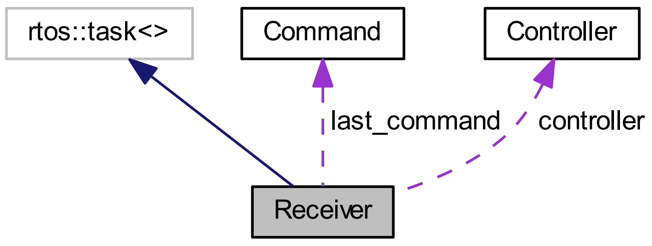
Class that handles receiving IR handling.

```
#include <receiver.h>
```

Inheritance diagram for Receiver:



Collaboration diagram for Receiver:



Public Member Functions

- Receiver (const char ∗name, hwlib::pin_in &signal, Controller ∗controller)

  Constructor for the Receiver class.
- void enable ()

  sets the enable flag
- void set_controller (Controller ∗c)

  swaps out the controller that the receiver talks to
- Controller ∗ get_controller ()

  returns the controller that the receiver is talking to
- void idle ()
- void signal_found ()

Private Member Functions

- void main ()

  RTOS main loop that if enabled will listen for signals If a command is received twice successfully it will call the current controller it's Received method with as parameter the received command.

Private Attributes

- hwlib::pin_in & signal

  Pin that receives the ir pulses.
- Controller ∗ controller

  Reference to a abstract controller class.
- rtos::flag enabled

  RTOS flag to check if the receiver is enabled.
- Command last_command

  Buffer for the last received command.
- int amount_bits_found = 0

  Buffer to keep track on how many bits we've counted for command.
- const int max_bits = 16

  Maximum size of the command bits.
- short bits

  a short in which the received bits will be shifted into

5.9.1 Detailed Description

Class that handles receiving IR handling.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 Receiver::Receiver (const char ∗ *name*, hwlib::pin_in & *signal*, Controller ∗ *controller*) [inline]

Constructor for the Receiver class.

Parameters

name	Task name
signal	Pin that's used to received to signals
controler	Reference to a abstract class controller

5.9.3 Member Function Documentation

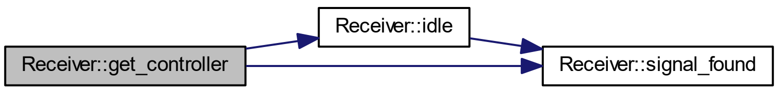
5.9.3.1 void Receiver::enable ( ) [inline]

sets the enable flag

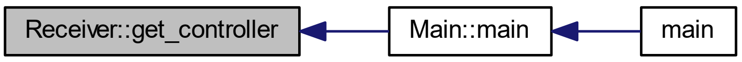
5.9.3.2 Controller* Receiver::get_controller ( ) [inline]

returns the controller that the receiver is talking to

Here is the call graph for this function:

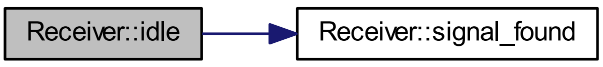


Here is the caller graph for this function:

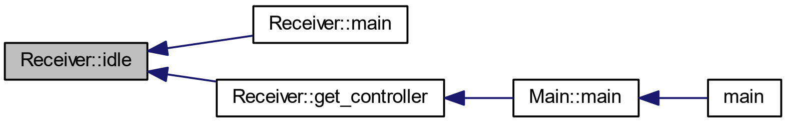


5.9.3.3 void Receiver::idle ( )

Here is the call graph for this function:



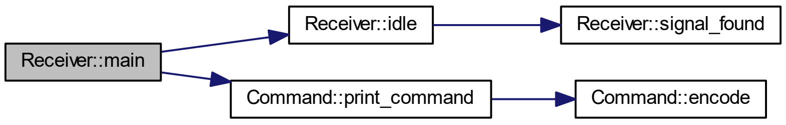
Here is the caller graph for this function:



5.9.3.4 void Receiver::main ( ) [private]

RTOS main loop that if enabled will listen for signals If a command is received twice successfully it will call the current controller it's Received method with as parameter the received command.

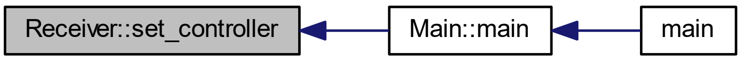
Here is the call graph for this function:



5.9.3.5 void Receiver::set_controller (Controller ∗ *c*) [inline]

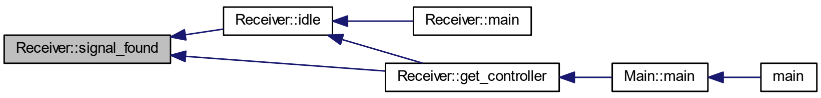
swaps out the controller that the receiver talks to

Here is the caller graph for this function:



5.9.3.6 void Receiver::signal_found ( )

Here is the caller graph for this function:



5.9.4 Member Data Documentation

5.9.4.1 int Receiver::amount_bits_found = 0 [private]

Buffer to keep track on how many bits we've counted for command.

5.9.4.2 short Receiver::bits [private]

a short in which the received bits will be shifted into

5.9.4.3 Controller* Receiver::controller [private]

Reference to a abstract controller class.

5.9.4.4 rtos::flag Receiver::enabled [private]

RTOS flag to check if the receiver is enabled.

5.9.4.5 Command Receiver::last_command [private]

Buffer for the last received command.

5.9.4.6 const int Receiver::max_bits = 16 [private]

Maximum size of the command bits.

5.9.4.7 hwlib::pin_in& Receiver::signal [private]

Pin that receives the ir pulses.

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

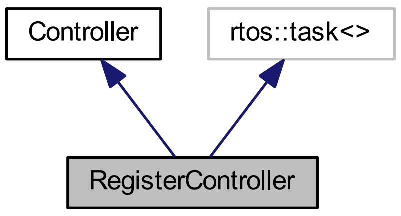
- src/tasks/receiver.h
- src/tasks/receiver.cpp

5.10 RegisterController Class Reference

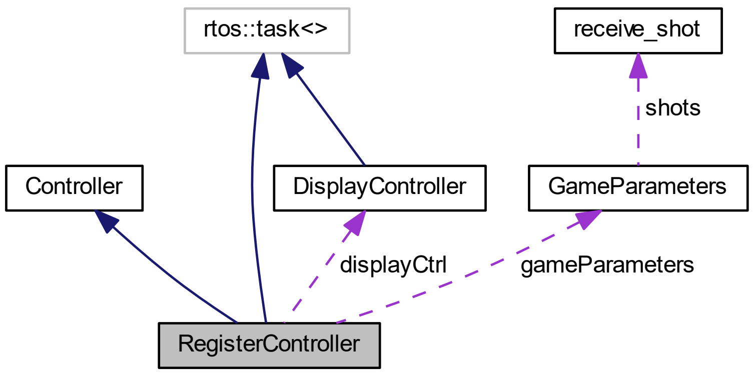
can be interpret as register state and will handle each event during his state.

```
#include <registerController.h>
```

Inheritance diagram for RegisterController:



Collaboration diagram for RegisterController:



Public Member Functions

- void enable ()
- void button_pressed ()
- void receive (Command c)
- const char ∗ get_name ()
- int state ()
- RegisterController (GameParameters &gp, DisplayController &dCtrl)

Private Member Functions

- void main ()

Private Attributes

- rtos::flag enabled
- rtos::flag pressed
- rtos::flag command_processed
- GameParameters & gameParameters
- DisplayController & displayCtrl
- bool ready_to_receive = false
- char next_state = 0

5.10.1 Detailed Description

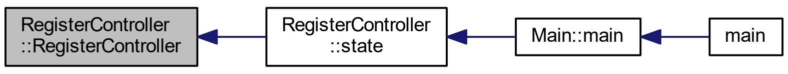
can be interpret as register state and will handle each event during his state.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 RegisterController::RegisterController (**GameParameters &** *gp,* **DisplayController &** *dCtrl*)

RegisterController Constructor /param GameParameters &gp for get and set player data /param Display Controller &dCtrl for get and set player data

Here is the caller graph for this function:


RegisterController ::RegisterController
RegisterController ::state
Main::main
main

5.10.3 Member Function Documentation

5.10.3.1 void RegisterController::button_pressed ( ) [virtual]

interface function to set button_pressed flag

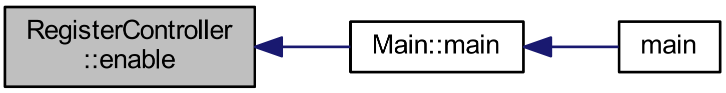
Implements Controller.

5.10.3.2 void RegisterController::enable ( ) [virtual]

interface function to activate task function

Implements Controller.

Here is the caller graph for this function:

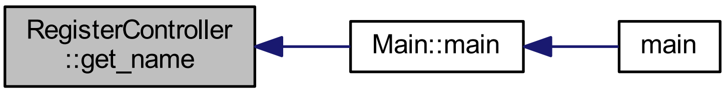


5.10.3.3 const char∗ RegisterController::get_name ( ) [inline], [virtual]

interface function to get name of the controller

Implements Controller.

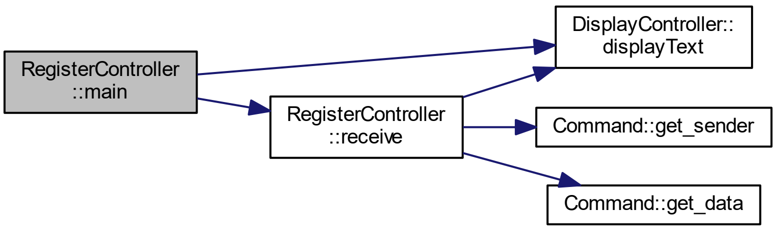
Here is the caller graph for this function:



5.10.3.4 void RegisterController::main ( ) [private]

RTOS main tas

Here is the call graph for this function:

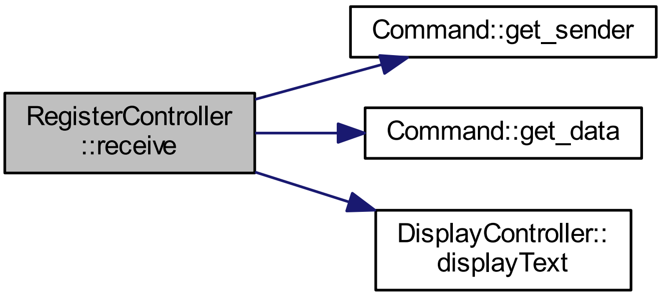

RegisterController ::main
RegisterController ::receive
DisplayController:: displayText
Command::get_sender
Command::get_data

5.10.3.5 void RegisterController::receive (**Command *c***) `[virtual]`

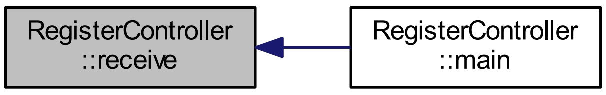
interface function for receiving data /param Command c to encode and decode data

Implements Controller.

Here is the call graph for this function:



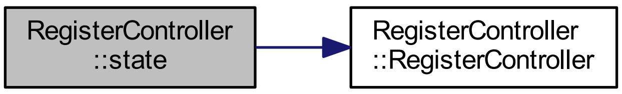
Here is the caller graph for this function:



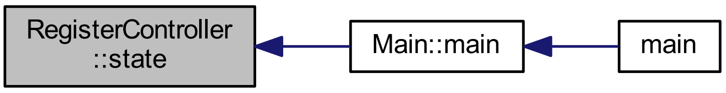
5.10.3.6 int RegisterController::state ( ) `[inline]`

function for change stat

Here is the call graph for this function:


RegisterController ::state
RegisterController ::RegisterController

Here is the caller graph for this function:



5.10.4 Member Data Documentation

5.10.4.1 rtos::flag RegisterController::command_processed [private]

command_available flag to check if command is available

5.10.4.2 DisplayController& RegisterController::displayCtrl [private]

displayCtrl for writing text oled

5.10.4.3 rtos::flag RegisterController::enabled [private]

enable flag to continue task

5.10.4.4 GameParameters& RegisterController::gameParameters [private]

gameParameters reference for get and set player data

5.10.4.5 char RegisterController::next_state = 0 [private]

char for changing state

5.10.4.6 rtos::flag RegisterController::pressed [private]

pressed flag to check for button presses

5.10.4.7 bool RegisterController::ready_to_receive = false [private]

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

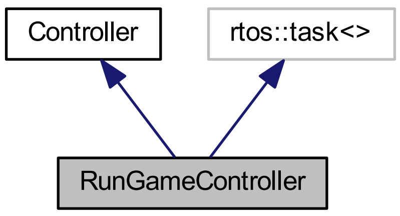
- src/tasks/registerController.h
- src/tasks/registerController.cpp

5.11 RunGameController Class Reference

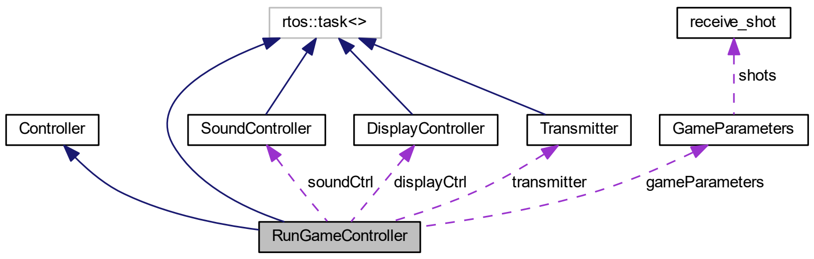
can be interpretted as playing state and will handle each event during his state.

```
#include <runGameController.h>
```

Inheritance diagram for RunGameController:



Collaboration diagram for RunGameController:



Public Member Functions

- void enable ()

  interface function for activating task.A
- void receive (Command c)

  interface function for receiving a command
- const char ∗ get_name ()

  interface function for getting controller name.
- void button_pressed ()

  button_pressed interface function.
- void update_screen_game_parameters (bool alive)

  Update the oled display with game timer and current health points.
- RunGameController (GameParameters &gameParam, DisplayController &disCtrl, Transmitter &t, Sound↩ Controller &sCtrl)

Private Member Functions

- void main ()

Private Attributes

- rtos::flag enabled

  fflag to enable the task
- rtos::flag pressed

  rtos flag will be used for detecting button press
- rtos::flag hit

  rtos flag will be used for detecting hits
- rtos::clock game_timer

  rtos clock that will count down the minutes
- GameParameters & gameParameters

  gameParameters will be an entity object for storring the personal player data.
- DisplayController & displayCtrl

  DisplayController reference handles text display on oled screen.
- Transmitter & transmitter

  Transmitter reference handles sending data over IR.
- SoundController & soundCtrl

  SoundController reference handles sound.

5.11.1 Detailed Description

can be interpretted as playing state and will handle each event during his state.

5.11.2 Constructor & Destructor Documentation

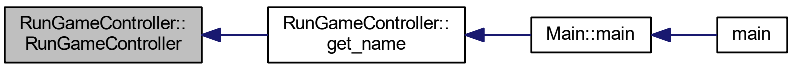
5.11.2.1 RunGameController::RunGameController (**GameParameters** & *gameParam*, **DisplayController** & *disCtrl*, **Transmitter** & *t*, **SoundController** & *sCtrl*)

RunGameController constructor.

Parameters

&gameParam	to set or get player game settings.
&disCtrl	to write text on screen.

Here is the caller graph for this function:


RunGameController:: RunGameController
RunGameController:: get_name
Main::main
main

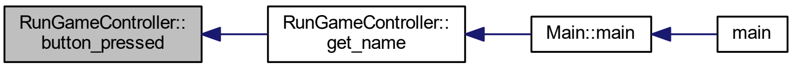
5.11.3 Member Function Documentation

5.11.3.1 void RunGameController::button_pressed ( ) [virtual]

button_pressed interface function.

Implements Controller.

Here is the caller graph for this function:

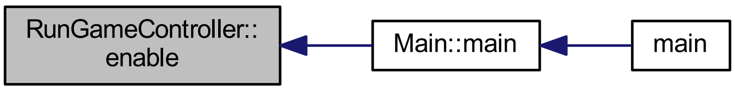


5.11.3.2 void RunGameController::enable ( ) [virtual]

interface function for activating task.A

Implements Controller.

Here is the caller graph for this function:

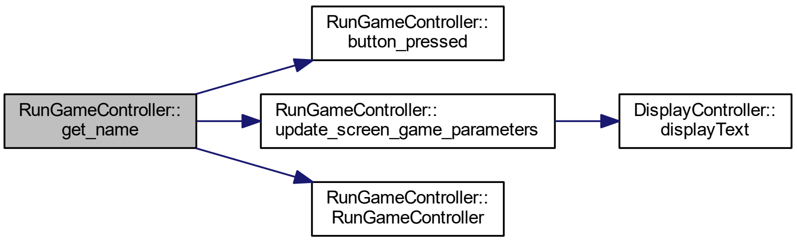


5.11.3.3 const char∗ RunGameController::get_name ( ) [inline], [virtual]

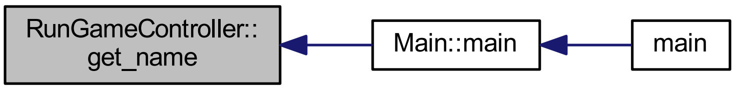
interface function for getting controller name.

Implements Controller.

Here is the call graph for this function:



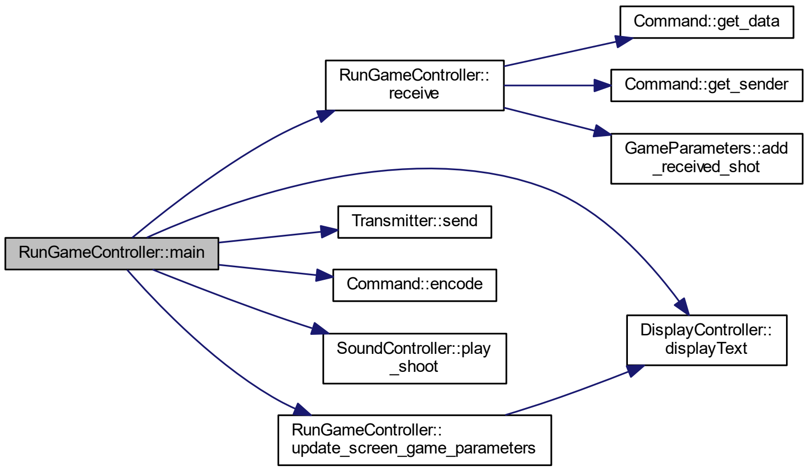
Here is the caller graph for this function:



5.11.3.4 void RunGameController::main ( ) [private]

task function.

Here is the call graph for this function:



5.11.3.5 void RunGameController::receive (**Command *c***) [virtual]

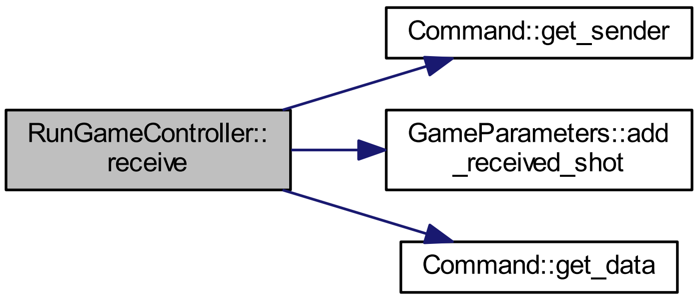
interface function for receiving a command

Parameters

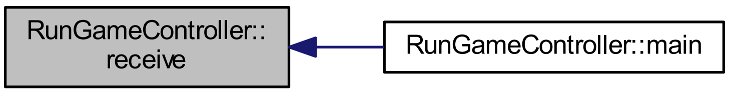
Command	c for encode en decode data.

Implements Controller.

Here is the call graph for this function:



Here is the caller graph for this function:



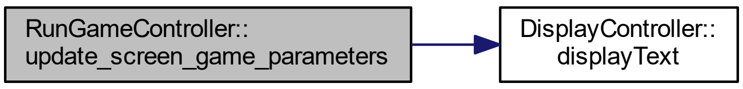
5.11.3.6 void RunGameController::update_screen_game_parameters (bool *alive*)

Update the oled display with game timer and current health points.

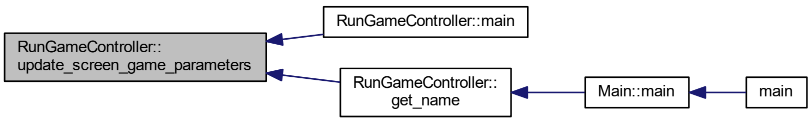
Parameters

alive	a simple boolean to show a alive display layout or dead layout q

Here is the call graph for this function:



Here is the caller graph for this function:



5.11.4 Member Data Documentation

5.11.4.1 DisplayController& RunGameController::displayCtrl `[private]`

DisplayController reference handles text display on oled screen.

5.11.4.2 rtos::flag RunGameController::enabled `[private]`

fflag to enable the task

5.11.4.3 rtos::clock RunGameController::game_timer `[private]`

rtos clock that will count down the minutes

5.11.4.4 GameParameters& RunGameController::gameParameters `[private]`

gameParameters will be an entity object for storring the personal player data.

5.11.4.5 rtos::flag RunGameController::hit `[private]`

rtos flag will be used for detecting hits

5.11.4.6 rtos::flag RunGameController::pressed `[private]`

rtos flag will be used for detecting button press

5.11.4.7 SoundController& RunGameController::soundCtrl `[private]`

SoundController reference handles sound.

5.11.4.8 Transmitter& RunGameController::transmitter `[private]`

Transmitter reference handles sending data over IR.

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

- src/tasks/runGameController.h
- src/tasks/runGameController.cpp

5.12 Sound Struct Reference

simple struct to maintain frequency and duration data

```
#include <soundController.h>
```

Public Attributes

- int frequency
- int duration

5.12.1 Detailed Description

simple struct to maintain frequency and duration data

5.12.2 Member Data Documentation

5.12.2.1 int Sound::duration

5.12.2.2 int Sound::frequency

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

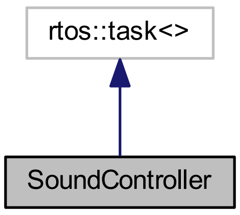
- src/tasks/soundController.h

5.13 SoundController Class Reference

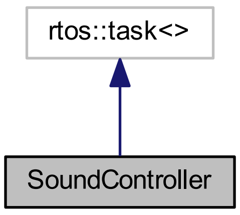
Simple task that can play a predefined sound.

```
#include <soundController.h>
```

Inheritance diagram for SoundController:



Collaboration diagram for SoundController:



Public Member Functions

- SoundController (hwlib::pin_out &lsp)

  Constructor of the SoundController.
- void play_shoot ()

  Simple function which will fill the sound channel with specific sounds. and set the play_sound flag so a sound will be played.

Private Member Functions

- void play (Sound s)

  function to translate frequency and duration into pulses
- void main ()

Private Attributes

- hwlib::pin_out & lsp
- rtos::flag play_sound
- rtos::channel< Sound, 20 > sounds

5.13.1 Detailed Description

Simple task that can play a predefined sound.

This class has the ability of playing a so called shoot sound

5.13.2 Constructor & Destructor Documentation

5.13.2.1 SoundController::SoundController (hwlib::pin_out & *lsp*)

Constructor of the SoundController.

Parameters

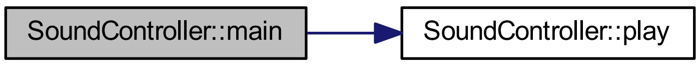
lsp	Pin that will be turn on and off to send pulses to the speaker

5.13.3 Member Function Documentation

5.13.3.1 void SoundController::main ( ) [private]

Rtos main loop Will wait on the flag play_sound to be set. If the flag is set it will read from the channel and play the sounds stored in the channel;

Here is the call graph for this function:



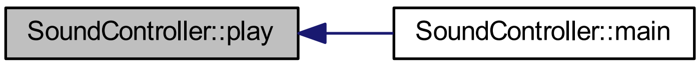
5.13.3.2 void SoundController::play (Sound *s*) [private]

function to translate frequency and duration into pulses

Parameters

s	A sound struct with frequency and duration defined

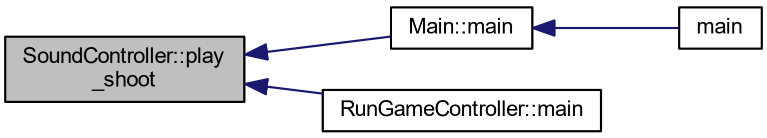
Here is the caller graph for this function:



5.13.3.3 void SoundController::play_shoot ( )

Simple function which will fill the sound channel with specific sounds. and set the play_sound flag so a sound will be played.

Here is the caller graph for this function:



5.13.4 Member Data Documentation

5.13.4.1 hwlib::pin_out& SoundController::lsp [private]

Pin used for outputting sound to the speaker

5.13.4.2 rtos::flag SoundController::play_sound [private]

Flag to be set if sound can be playe

5.13.4.3 rtos::channel<Sound, 20> SoundController::sounds [private]

Simple channel which will hold a wave of sounds

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

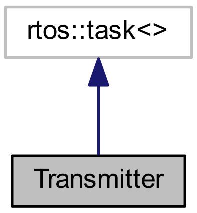
- src/tasks/soundController.h
- src/tasks/soundController.cpp

5.14 Transmitter Class Reference

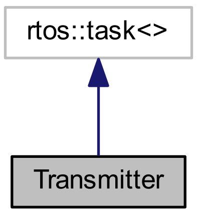
Tranmitter class used to send information over IR A rtos task that gives the user the ability of sending a SHORT through ir in binary form.

```
#include <transmitter.h>
```

Inheritance diagram for Transmitter:



Collaboration diagram for Transmitter:



Public Member Functions

- Transmitter (const char ∗name, hwlib::target::d2_36kHz &ir)

    Constructor for the Transmitter.
- void send (short bits)

    Send given short in binary msb over IR.

Private Member Functions

- void main ()

Private Attributes

- hwlib::target::d2_36kHz & ir
- rtos::flag command_received
- short command_bits

5.14.1 Detailed Description

Tranmitter class used to send information over IR A rtos task that gives the user the ability of sending a SHORT through ir in binary form.

5.14.2 Constructor & Destructor Documentation

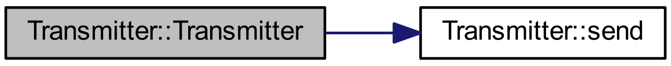
5.14.2.1 Transmitter::Transmitter (const char ∗ *name,* hwlib::target::d2_36kHz & *ir*) [inline]

Constructor for the Transmitter.

Parameters

name	Name of the task
ir	Reference to already instantiated d2_36khz class

Here is the call graph for this function:



5.14.3 Member Function Documentation

5.14.3.1 void Transmitter::main ( ) [private]

Rtos main loop that sends a short in binary form over IR using the d2_36khz pin

5.14.3.2 void Transmitter::send (short *bits*)

Send given short in binary msb over IR.

Uses the defined protocol of the project It sends a 0 by sending for 800us and silent for 1600us

It sends a 1 by sending for 1600us and silent for 800us

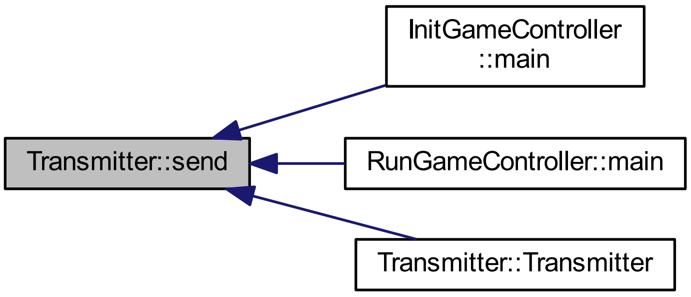
The command is sent twice in a delay of 3ms

The starbit is a simple 1

Parameters

bits	The short that will be send by binary MSB over ir

Here is the caller graph for this function:



5.14.4 Member Data Documentation

5.14.4.1 short Transmitter::command_bits [private]

short to save the command in

5.14.4.2 rtos::flag Transmitter::command_received [private]

An RTOS flag used to check if a command is received

5.14.4.3 hwlib::target::d2_36kHz& Transmitter::ir [private]

Reference to a instance of a hwlib::target::d2_36kHz class

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

- src/tasks/transmitter.h
- src/tasks/transmitter.cpp

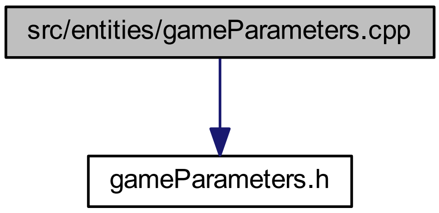
Chapter 6

File Documentation

6.1 src/entities/gameParameters.cpp File Reference

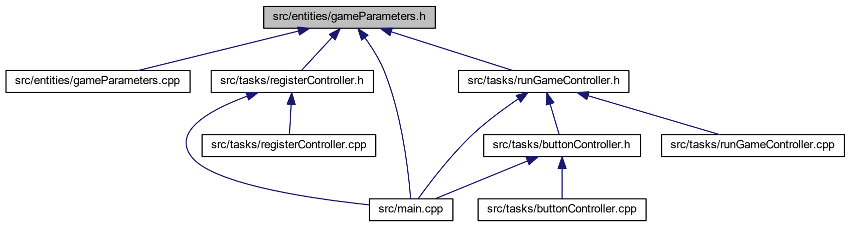
```
#include "gameParameters.h"
```

Include dependency graph for gameParameters.cpp:



6.2 src/entities/gameParameters.h File Reference

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


src/entities/gameParameters.h
src/entities/gameParameters.cpp
src/tasks/registerController.h
src/tasks/runGameController.h
src/tasks/registerController.cpp
src/tasks/buttonController.h
src/tasks/runGameController.cpp
src/main.cpp
src/tasks/buttonController.cpp

Classes

- struct receive_shot

  receive_shot structure. Will be used for class GameParameters. Each structure contains the player id and weapon id from the received hit

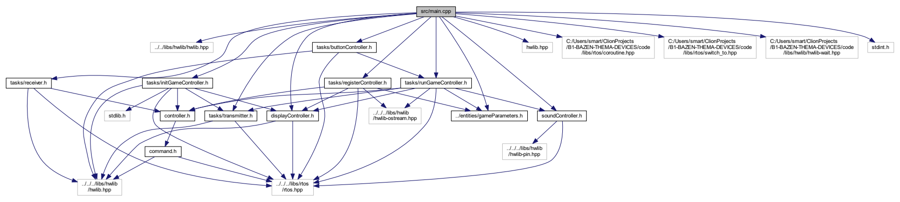
- class GameParameters

  GameParameters entity object will contain the player data.

6.3 src/main.cpp File Reference

```
#include "../../libs/hwlib/hwlib.hpp"
#include "tasks/transmitter.h"
#include "tasks/receiver.h"
#include "tasks/initGameController.h"
#include "tasks/registerController.h"
#include "tasks/displayController.h"
#include "entities/gameParameters.h"
#include "tasks/runGameController.h"
#include "tasks/buttonController.h"
#include "tasks/soundController.h"
```

Include dependency graph for main.cpp:


src/main.cpp
../../libs/hwlib/hwlib.hpp
tasks/buttonController.h
hwlib.hpp
C:/Users/smart/ClionProjects/B1-BAZEN-THEMA-DEVICES/code/libs/rtos/coroutine.hpp
C:/Users/smart/ClionProjects/B1-BAZEN-THEMA-DEVICES/code/libs/rtos/switch_to.hpp
C:/Users/smart/ClionProjects/B1-BAZEN-THEMA-DEVICES/code/libs/hwlib/hwlib-wait.hpp
stdint.h
tasks/receiver.h
tasks/initGameController.h
tasks/registerController.h
tasks/runGameController.h
stdlib.h
controller.h
tasks/transmitter.h
displayController.h
../../../libs/hwlib/hwlib-ostream.hpp
../entities/gameParameters.h
soundController.h
../../../libs/hwlib/hwlib-pin.hpp
command.h
../../../libs/hwlib/hwlib.hpp
../../../libs/rtos/rtos.hpp

Classes

- class Main

  This class will resume and suspend running tasks based on it's current state.

Enumerations

- enum States { INIT, REGISTER, RUNNING, GAME_END }

Functions

- int main ()

Variables

- States current_state = REGISTER

6.3.1 Enumeration Type Documentation

6.3.1.1 enum States

Enumerator

INIT

REGISTER

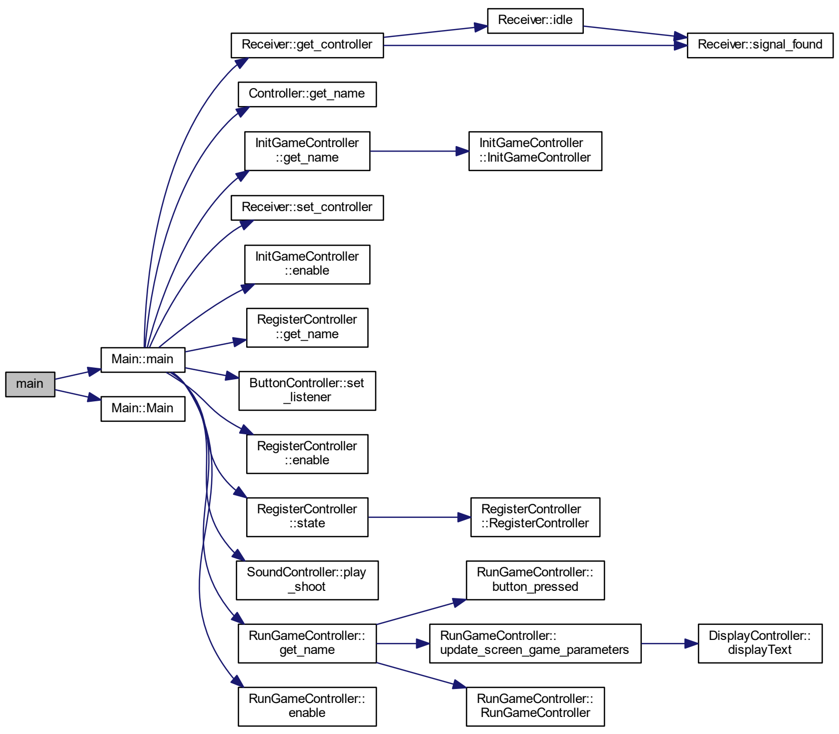
RUNNING

GAME_END

6.3.2 Function Documentation

6.3.2.1 int main ( )

Here is the call graph for this function:

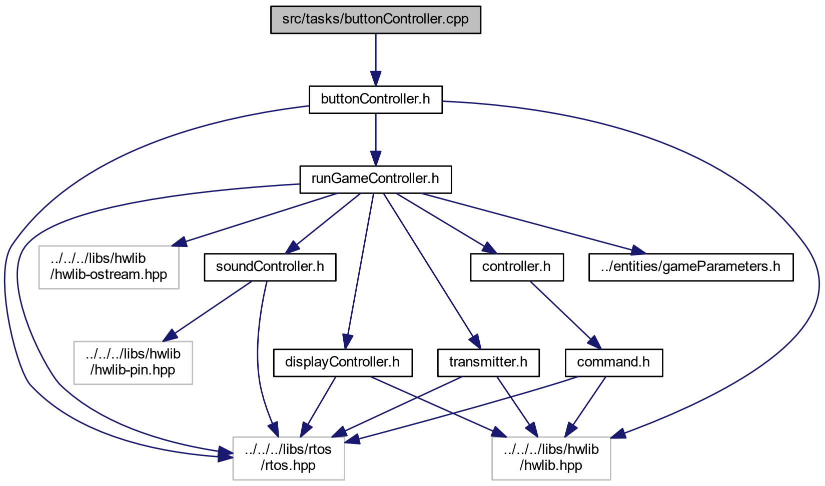

main
Main::main
Main::Main
Receiver::get_controller
Receiver::idle
Receiver::signal_found
Controller::get_name
InitGameController ::get_name
InitGameController ::InitGameController
Receiver::set_controller
InitGameController ::enable
RegisterController ::get_name
ButtonController::set _listener
RegisterController ::enable
RegisterController ::state
RegisterController ::RegisterController
SoundController::play _shoot
RunGameController:: button_pressed
RunGameController:: get_name
RunGameController:: update_screen_game_parameters
DisplayController:: displayText
RunGameController:: enable
RunGameController:: RunGameController

6.3.3 Variable Documentation

6.3.3.1 States current_state = REGISTER

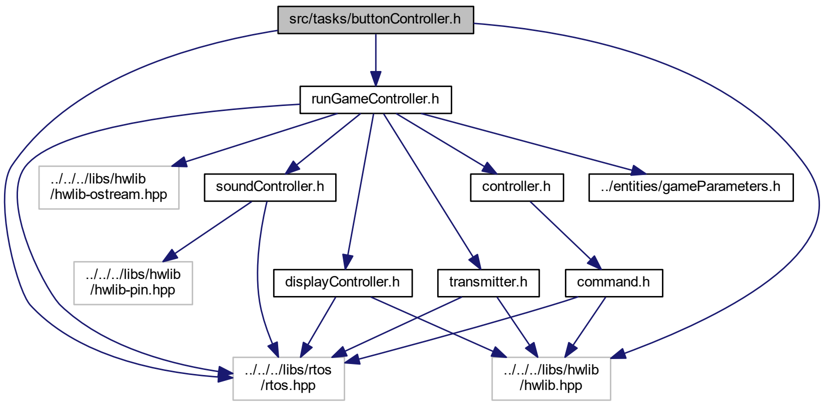
6.4 src/tasks/buttonController.cpp File Reference

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

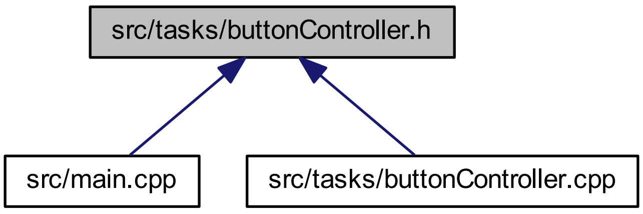

src/tasks/buttonController.cpp
buttonController.h
runGameController.h
../../../libs/hwlib/hwlib-ostream.hpp
soundController.h
controller.h
../entities/gameParameters.h
../../../libs/hwlib/hwlib-pin.hpp
displayController.h
transmitter.h
command.h
../../../libs/rtos/rtos.hpp
../../../libs/hwlib/hwlib.hpp

6.5 src/tasks/buttonController.h File Reference

```
#include "../../../libs/rtos/rtos.hpp"
#include "../../../libs/hwlib/hwlib.hpp"
#include "runGameController.h"
```
Include dependency graph for buttonController.h:


src/tasks/buttonController.h
runGameController.h
../../../libs/hwlib/hwlib-ostream.hpp
soundController.h
controller.h
../entities/gameParameters.h
../../../libs/hwlib/hwlib-pin.hpp
displayController.h
transmitter.h
command.h
../../../libs/rtos/rtos.hpp
../../../libs/hwlib/hwlib.hpp

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



Classes

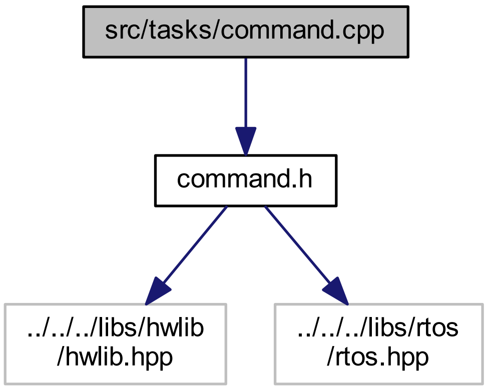
- class ButtonController

  can be interpreted as playing state and will handle each event during his state.

6.6 src/tasks/command.cpp File Reference

```
#include "command.h"
```

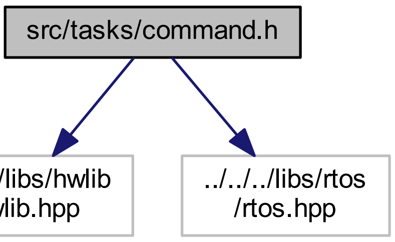
Include dependency graph for command.cpp:



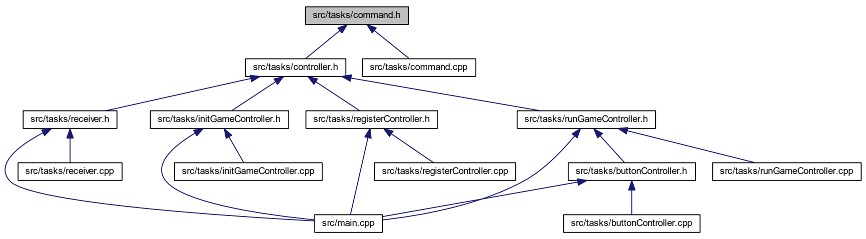
6.7 src/tasks/command.h File Reference

```
#include "../../../libs/hwlib/hwlib.hpp"
#include "../../../libs/rtos/rtos.hpp"
```

Include dependency graph for command.h:



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


src/tasks/command.h
src/tasks/controller.h
src/tasks/command.cpp
src/tasks/receiver.h
src/tasks/initGameController.h
src/tasks/registerController.h
src/tasks/runGameController.h
src/tasks/receiver.cpp
src/tasks/initGameController.cpp
src/tasks/registerController.cpp
src/tasks/buttonController.h
src/tasks/runGameController.cpp
src/main.cpp
src/tasks/buttonController.cpp

Classes

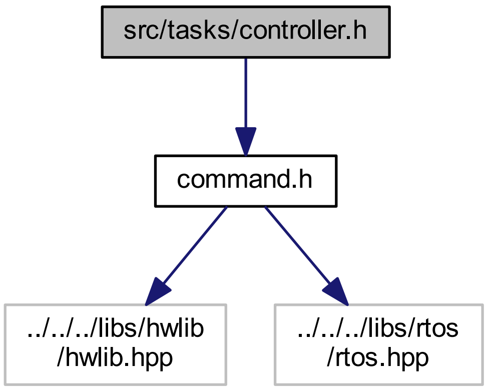
- class Command

  Command that handles decoding and encoding of the IR commands This class can be instantiated with a short If this happens it will automatically decode the short into readable data like the sender and the actual data.

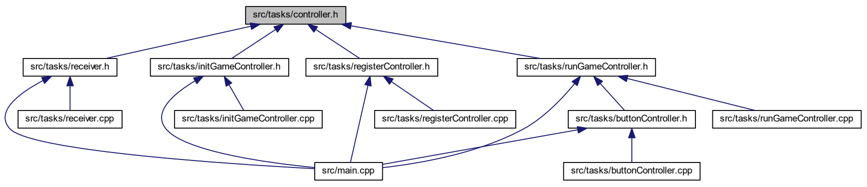
6.8 src/tasks/controller.h File Reference

```
#include "command.h"
```

Include dependency graph for controller.h:



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


src/tasks/controller.h
src/tasks/receiver.h
src/tasks/initGameController.h
src/tasks/registerController.h
src/tasks/runGameController.h
src/tasks/receiver.cpp
src/tasks/initGameController.cpp
src/tasks/registerController.cpp
src/tasks/buttonController.h
src/tasks/runGameController.cpp
src/main.cpp
src/tasks/buttonController.cpp

Classes

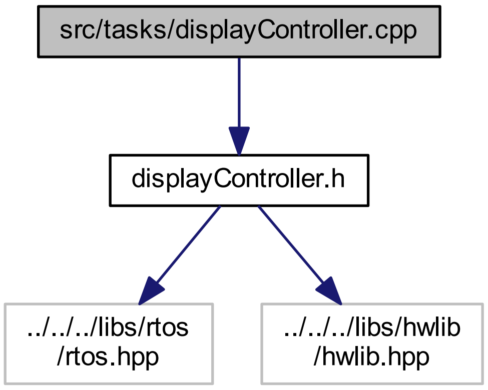
- class Controller

  will be implented by each state

6.9 src/tasks/displayController.cpp File Reference

```
#include "displayController.h"
```

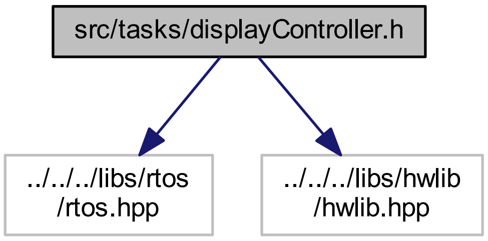
Include dependency graph for displayController.cpp:


src/tasks/displayController.cpp
displayController.h
../../../libs/rtos/rtos.hpp
../../../libs/hwlib/hwlib.hpp

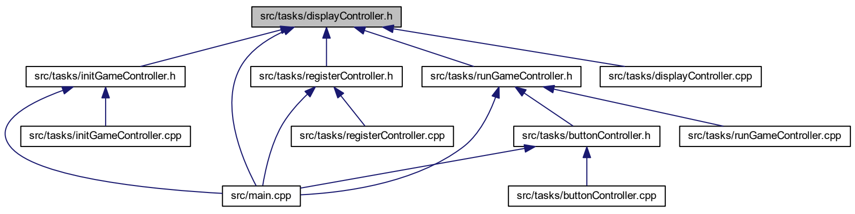
6.10 src/tasks/displayController.h File Reference

```
#include "../../../libs/rtos/rtos.hpp"
#include "../../../libs/hwlib/hwlib.hpp"
```

Include dependency graph for displayController.h:


src/tasks/displayController.h
../../../libs/rtos/rtos.hpp
../../../libs/hwlib/hwlib.hpp

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


src/tasks/displayController.h
src/tasks/initGameController.h
src/tasks/registerController.h
src/tasks/runGameController.h
src/tasks/displayController.cpp
src/tasks/initGameController.cpp
src/tasks/registerController.cpp
src/tasks/buttonController.h
src/tasks/runGameController.cpp
src/main.cpp
src/tasks/buttonController.cpp

Classes

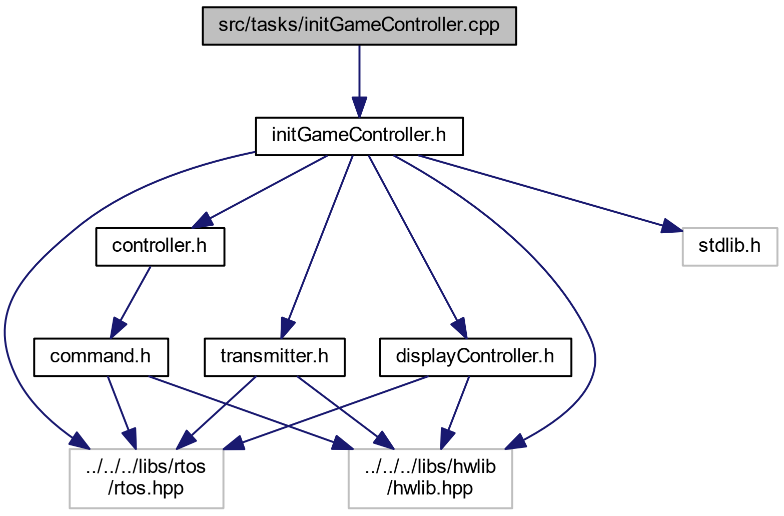
- class DisplayController

    will be used as communcation controller between oled boundary and other game state controllers

6.11 src/tasks/initGameController.cpp File Reference

```
#include "initGameController.h"
```

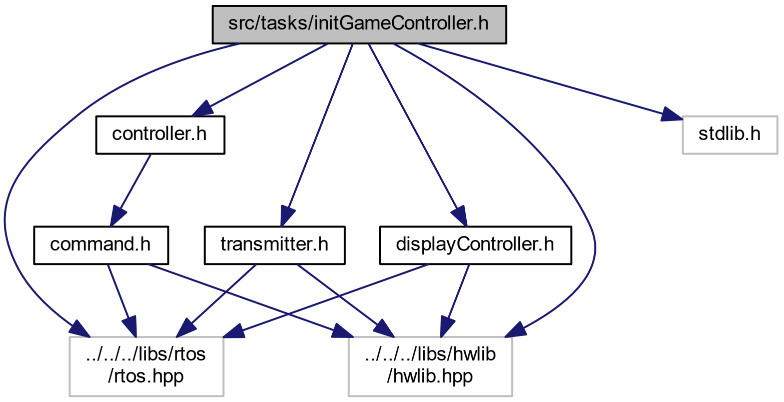
Include dependency graph for initGameController.cpp:



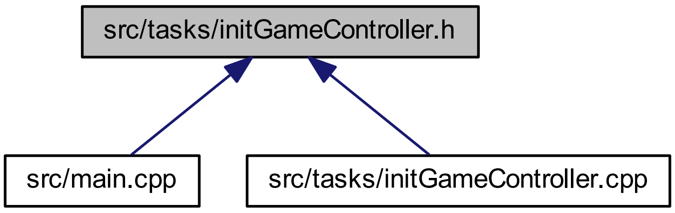
6.12 src/tasks/initGameController.h File Reference

```
#include "../../../libs/rtos/rtos.hpp"
#include "../../../libs/hwlib/hwlib.hpp"
#include "controller.h"
#include "transmitter.h"
#include "displayController.h"
#include "stdlib.h"
```

Include dependency graph for initGameController.h:



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


src/tasks/initGameController.h
src/main.cpp
src/tasks/initGameController.cpp

Classes

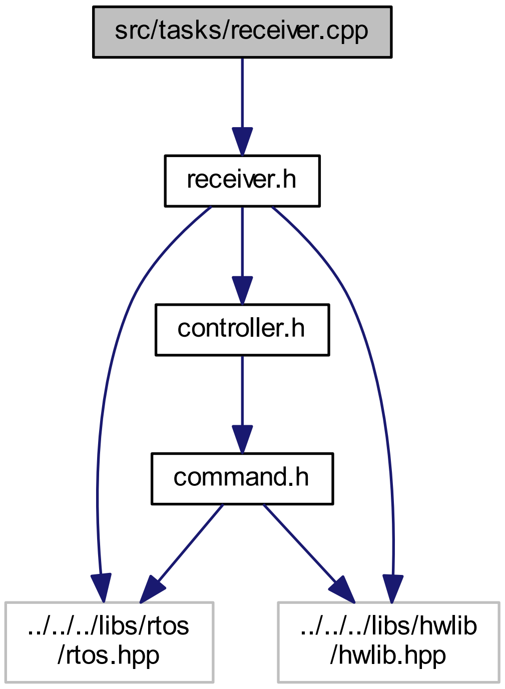
- class InitGameController

  can be interpretted as playing state and will handle each event during his state.

6.13 src/tasks/receiver.cpp File Reference

```
#include "receiver.h"
```

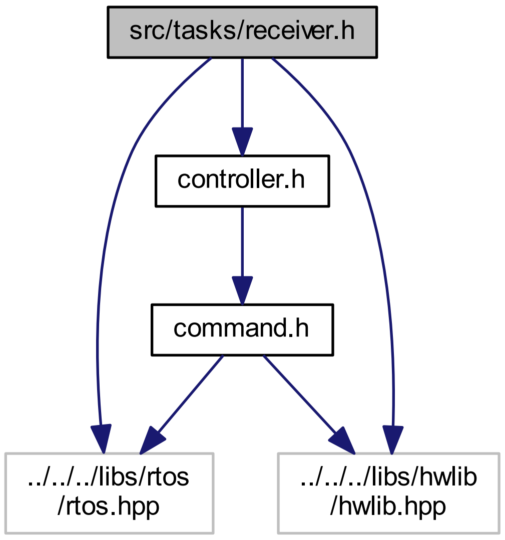
Include dependency graph for receiver.cpp:



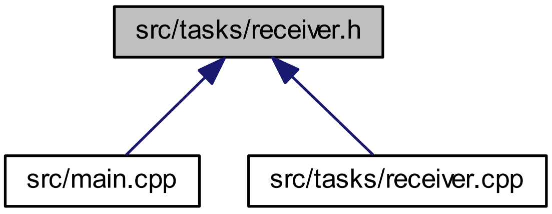
6.14 src/tasks/receiver.h File Reference

```
#include "../../../libs/rtos/rtos.hpp"
#include "../../../libs/hwlib/hwlib.hpp"
#include "controller.h"
```

Include dependency graph for receiver.h:



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



Classes

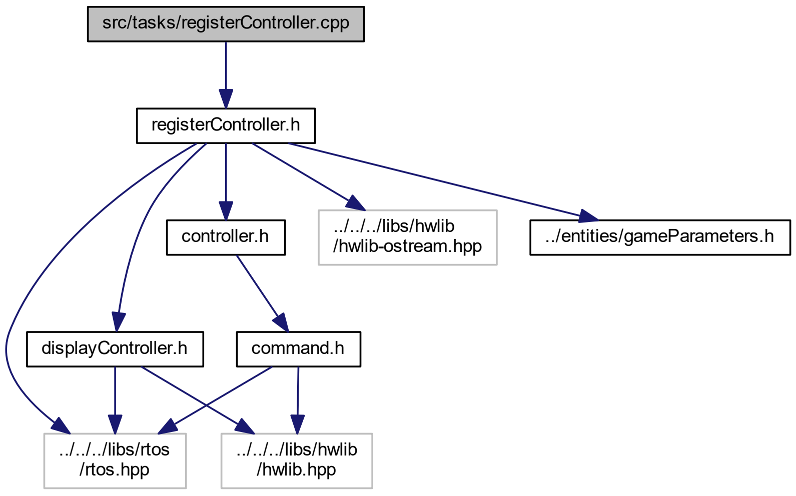
- class Receiver

    Class that handles receiving IR handling.

6.15 src/tasks/registerController.cpp File Reference

```
#include "registerController.h"
```

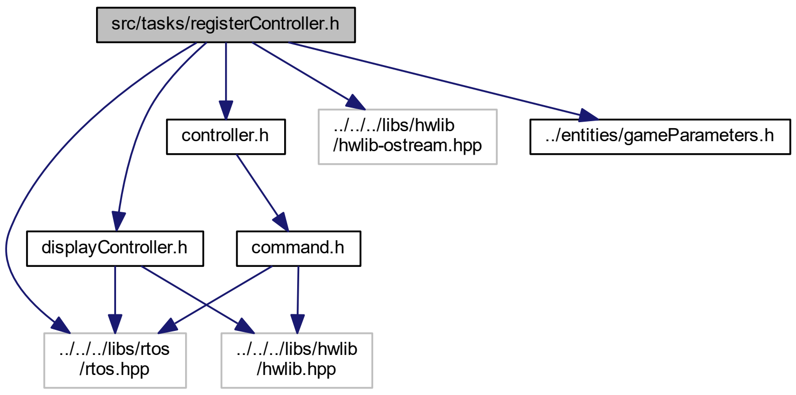
Include dependency graph for registerController.cpp:



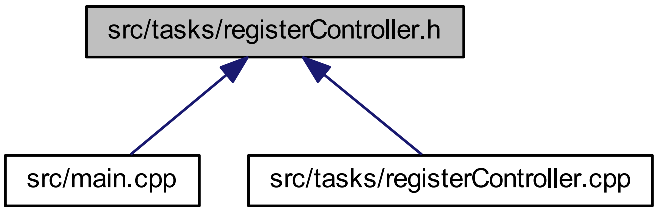
6.16 src/tasks/registerController.h File Reference

```
#include "../../../libs/rtos/rtos.hpp"
#include "controller.h"
#include "../../../libs/hwlib/hwlib-ostream.hpp"
#include "displayController.h"
#include "../entities/gameParameters.h"
```

Include dependency graph for registerController.h:



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


src/tasks/registerController.h
src/main.cpp
src/tasks/registerController.cpp

Classes

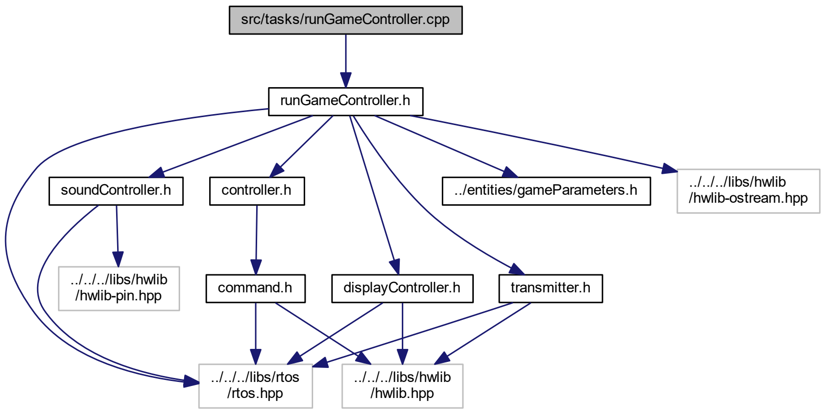
- class RegisterController

  can be interpret as register state and will handle each event during his state.

6.17 src/tasks/runGameController.cpp File Reference

```
#include "runGameController.h"
```

Include dependency graph for runGameController.cpp:

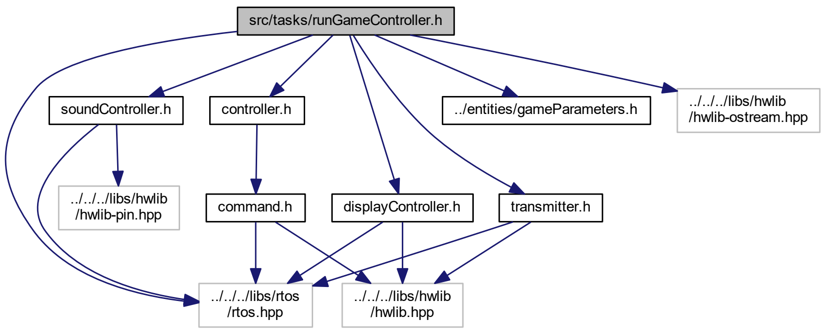

src/tasks/runGameController.cpp
runGameController.h
soundController.h
controller.h
../entities/gameParameters.h
../../../libs/hwlib /hwlib-ostream.hpp
../../../libs/hwlib /hwlib-pin.hpp
command.h
displayController.h
transmitter.h
../../../libs/rtos /rtos.hpp
../../../libs/hwlib /hwlib.hpp

6.18 src/tasks/runGameController.h File Reference

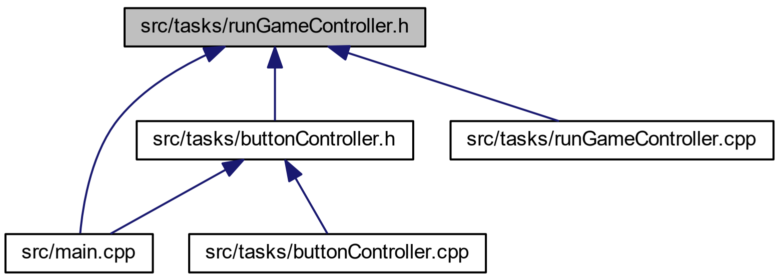
```
#include "../../../libs/rtos/rtos.hpp"
```

```
#include "../entities/gameParameters.h"
#include "controller.h"
#include "../../../libs/hwlib/hwlib-ostream.hpp"
#include "displayController.h"
#include "soundController.h"
#include "transmitter.h"
```

Include dependency graph for runGameController.h:


src/tasks/runGameController.h
soundController.h
controller.h
../entities/gameParameters.h
../../../libs/hwlib /hwlib-ostream.hpp
../../../libs/hwlib /hwlib-pin.hpp
command.h
displayController.h
transmitter.h
../../../libs/rtos /rtos.hpp
../../../libs/hwlib /hwlib.hpp

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


src/tasks/runGameController.h
src/tasks/buttonController.h
src/tasks/runGameController.cpp
src/main.cpp
src/tasks/buttonController.cpp

Classes

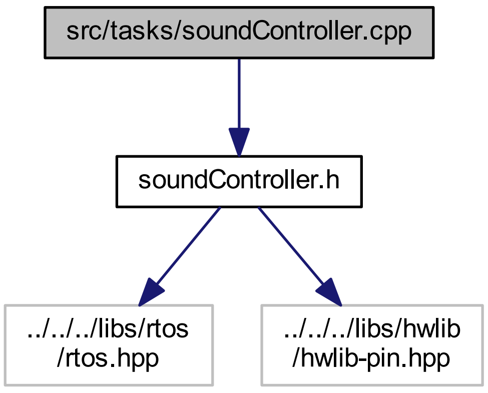
- class RunGameController

  can be interpretted as playing state and will handle each event during his state.

6.19 src/tasks/soundController.cpp File Reference

```
#include "soundController.h"
```

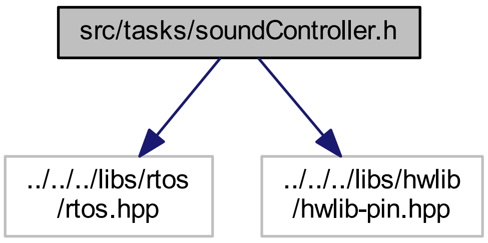
Include dependency graph for soundController.cpp:


src/tasks/soundController.cpp
soundController.h
../../../libs/rtos/rtos.hpp
../../../libs/hwlib/hwlib-pin.hpp

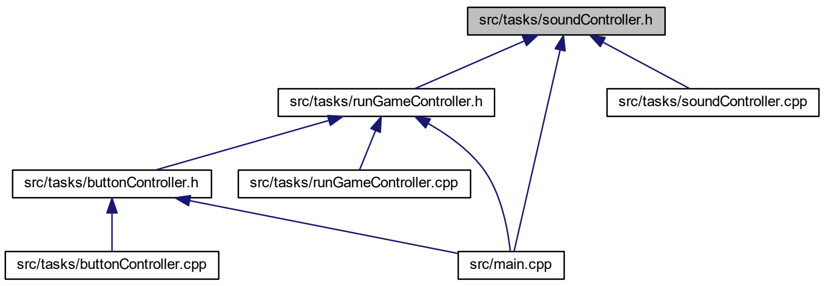
6.20 src/tasks/soundController.h File Reference

```
#include "../../../libs/rtos/rtos.hpp"
#include "../../../libs/hwlib/hwlib-pin.hpp"
```

Include dependency graph for soundController.h:


src/tasks/soundController.h
../../../libs/rtos/rtos.hpp
../../../libs/hwlib/hwlib-pin.hpp

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


src/tasks/soundController.h
src/tasks/runGameController.h
src/tasks/soundController.cpp
src/tasks/buttonController.h
src/tasks/runGameController.cpp
src/tasks/buttonController.cpp
src/main.cpp

Classes

- struct Sound

    simple struct to maintain frequency and duration data

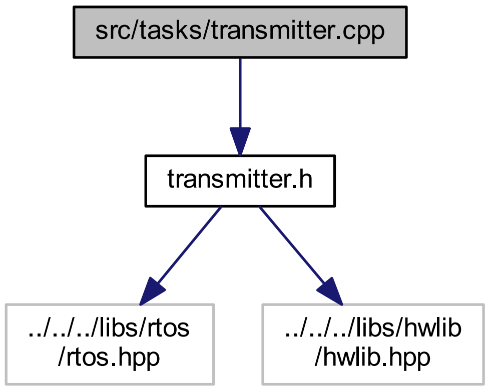
- class SoundController

    Simple task that can play a predefined sound.

6.21 src/tasks/transmitter.cpp File Reference

```
#include "transmitter.h"
```

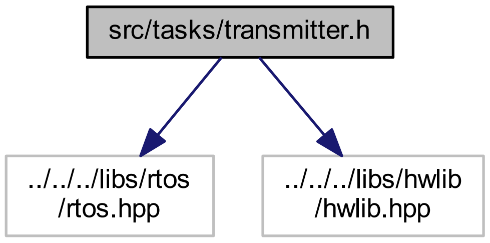
Include dependency graph for transmitter.cpp:



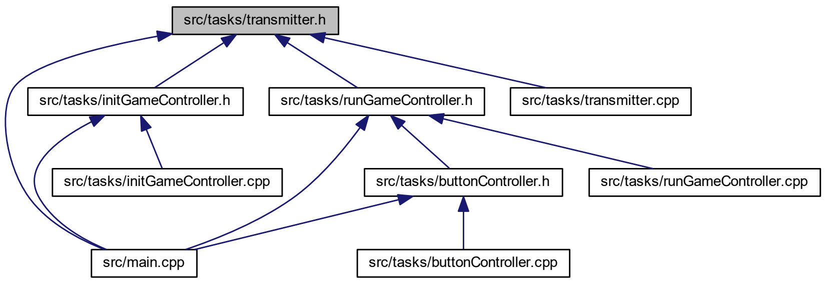
6.22 src/tasks/transmitter.h File Reference

```
#include "../../../libs/rtos/rtos.hpp"
#include "../../../libs/hwlib/hwlib.hpp"
```

Include dependency graph for transmitter.h:



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


src/tasks/transmitter.h
src/tasks/initGameController.h
src/tasks/runGameController.h
src/tasks/transmitter.cpp
src/tasks/initGameController.cpp
src/tasks/buttonController.h
src/tasks/runGameController.cpp
src/main.cpp
src/tasks/buttonController.cpp

Classes

- class Transmitter

  Tranmitter class used to send information over IR A rtos task that gives the user the ability of sending a SHORT through ir in binary form.

Index