

B1 bazen Laser Tag

1

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>Laser tag</b>	<b>1</b>
<b>2</b>	<b>Hierarchical Index</b>	<b>3</b>
2.1	Class Hierarchy . . . . .	3
<b>3</b>	<b>Class Index</b>	<b>5</b>
3.1	Class List . . . . .	5
<b>4</b>	<b>File Index</b>	<b>7</b>
4.1	File List . . . . .	7
<b>5</b>	<b>Class Documentation</b>	<b>9</b>
5.1	ButtonController Class Reference . . . . .	9
5.1.1	Detailed Description . . . . .	10
5.1.2	Constructor & Destructor Documentation . . . . .	10
5.1.2.1	ButtonController(Controller *ctrl, hwlib::pin_out &gnd, hwlib::pin_out &vlt, hwlib::pin_in &b) . . . . .	10
5.1.3	Member Function Documentation . . . . .	10
5.1.3.1	main() . . . . .	10
5.1.3.2	set_listener(Controller *ctrl) . . . . .	11
5.1.4	Member Data Documentation . . . . .	11
5.1.4.1	button . . . . .	11
5.1.4.2	clock . . . . .	11
5.1.4.3	controller . . . . .	11
5.1.4.4	ground . . . . .	11
5.1.4.5	voltage . . . . .	11

5.2	Command Class Reference	11
5.2.1	Detailed Description	12
5.2.2	Constructor & Destructor Documentation	13
5.2.2.1	Command()	13
5.2.2.2	Command(short bits)	13
5.2.2.3	Command(int sender, int data)	13
5.2.3	Member Function Documentation	14
5.2.3.1	decode(short bits)	14
5.2.3.2	encode()	14
5.2.3.3	get_data()	15
5.2.3.4	get_error()	15
5.2.3.5	get_sender()	15
5.2.3.6	print_command()	16
5.2.3.7	set_data(int data)	16
5.2.3.8	set_sender(int sender)	16
5.2.3.9	valid_checksum(short bits)	16
5.2.4	Member Data Documentation	17
5.2.4.1	data	17
5.2.4.2	error	17
5.2.4.3	sender	17
5.3	Controller Class Reference	17
5.3.1	Detailed Description	18
5.3.2	Constructor & Destructor Documentation	18
5.3.2.1	Controller()	18
5.3.3	Member Function Documentation	18
5.3.3.1	button_pressed()=0	18
5.3.3.2	enable()=0	19
5.3.3.3	get_name()=0	19
5.3.3.4	receive(Command c)=0	19
5.4	DisplayController Class Reference	20

5.4.1	Detailed Description	21
5.4.2	Constructor & Destructor Documentation	21
5.4.2.1	DisplayController(hwlib::glcd_oled_buffered &o)	21
5.4.3	Member Function Documentation	21
5.4.3.1	displayText(const char *)	21
5.4.3.2	main()	21
5.4.4	Member Data Documentation	21
5.4.4.1	buffer	21
5.4.4.2	clearFlag	21
5.4.4.3	flushFlag	22
5.4.4.4	oled	22
5.4.4.5	timer_screen	22
5.5	GameParameters Class Reference	22
5.5.1	Detailed Description	23
5.5.2	Constructor & Destructor Documentation	23
5.5.2.1	GameParameters()	23
5.5.3	Member Function Documentation	23
5.5.3.1	add_received_shot(int player_id, int weapon_id)	23
5.5.4	Member Data Documentation	23
5.5.4.1	game_time	23
5.5.4.2	health	24
5.5.4.3	id	24
5.5.4.4	shots	24
5.5.4.5	shots_taken	24
5.5.4.6	weapon	24
5.6	InitGameController Class Reference	24
5.6.1	Detailed Description	25
5.6.2	Constructor & Destructor Documentation	26
5.6.2.1	InitGameController(Transmitter &transmitter, hwlib::keypad< 16 > &keypad, DisplayController &displayCtrl)	26
5.6.3	Member Function Documentation	26

5.6.3.1	<a href="#">button_pressed()</a>	26
5.6.3.2	<a href="#">enable()</a>	26
5.6.3.3	<a href="#">get_name()</a>	27
5.6.3.4	<a href="#">main()</a>	27
5.6.3.5	<a href="#">receive(Command c)</a>	28
5.6.3.6	<a href="#">valid_id(char first, char second)</a>	28
5.6.4	<a href="#">Member Data Documentation</a>	29
5.6.4.1	<a href="#">command</a>	29
5.6.4.2	<a href="#">command_available</a>	29
5.6.4.3	<a href="#">command_full</a>	29
5.6.4.4	<a href="#">custom_command</a>	29
5.6.4.5	<a href="#">displayCtrl</a>	29
5.6.4.6	<a href="#">enabled</a>	29
5.6.4.7	<a href="#">keypad</a>	29
5.6.4.8	<a href="#">player_id</a>	29
5.6.4.9	<a href="#">transmitter</a>	29
5.6.4.10	<a href="#">weapon_id</a>	30
5.7	<a href="#">Main Class Reference</a>	30
5.7.1	<a href="#">Detailed Description</a>	32
5.7.2	<a href="#">Constructor &amp; Destructor Documentation</a>	32
5.7.2.1	<a href="#">Main(Receiver &amp;r, ButtonController &amp;b, InitGameController &amp;i, RegisterController &amp;reg, RunGameController &amp;run, SoundController &amp;sound)</a>	32
5.7.3	<a href="#">Member Function Documentation</a>	32
5.7.3.1	<a href="#">main()</a>	32
5.7.4	<a href="#">Member Data Documentation</a>	33
5.7.4.1	<a href="#">button_controller</a>	33
5.7.4.2	<a href="#">init_controller</a>	33
5.7.4.3	<a href="#">receiver</a>	34
5.7.4.4	<a href="#">register_controller</a>	34
5.7.4.5	<a href="#">run_game_controller</a>	34
5.7.4.6	<a href="#">sound_controller</a>	34

5.8	receive_shot Struct Reference	34
5.8.1	Detailed Description	34
5.8.2	Member Data Documentation	34
5.8.2.1	player_id	34
5.8.2.2	weapon_id	35
5.9	Receiver Class Reference	35
5.9.1	Detailed Description	36
5.9.2	Constructor & Destructor Documentation	36
5.9.2.1	Receiver(const char *name, hwlib::pin_in &signal, Controller *controller)	36
5.9.3	Member Function Documentation	37
5.9.3.1	enable()	37
5.9.3.2	get_controller()	37
5.9.3.3	idle()	37
5.9.3.4	main()	38
5.9.3.5	set_controller(Controller *c)	38
5.9.3.6	signal_found()	39
5.9.4	Member Data Documentation	39
5.9.4.1	amount_bits_found	39
5.9.4.2	bits	39
5.9.4.3	controller	39
5.9.4.4	enabled	39
5.9.4.5	last_command	39
5.9.4.6	max_bits	39
5.9.4.7	signal	39
5.10	RegisterController Class Reference	40
5.10.1	Detailed Description	41
5.10.2	Constructor & Destructor Documentation	41
5.10.2.1	RegisterController(GameParameters &gp, DisplayController &dCtrl)	41
5.10.3	Member Function Documentation	41
5.10.3.1	button_pressed()	41

5.10.3.2	<code>enable()</code>	42
5.10.3.3	<code>get_name()</code>	42
5.10.3.4	<code>main()</code>	42
5.10.3.5	<code>receive(Command c)</code>	43
5.10.3.6	<code>state()</code>	43
5.10.4	Member Data Documentation	44
5.10.4.1	<code>command_processed</code>	44
5.10.4.2	<code>displayCtrl</code>	44
5.10.4.3	<code>enabled</code>	44
5.10.4.4	<code>gameParameters</code>	44
5.10.4.5	<code>next_state</code>	44
5.10.4.6	<code>pressed</code>	44
5.10.4.7	<code>ready_to_receive</code>	44
5.11	RunGameController Class Reference	45
5.11.1	Detailed Description	46
5.11.2	Constructor & Destructor Documentation	46
5.11.2.1	<code>RunGameController(GameParameters &amp;gameParam, DisplayController &amp;disCtrl, Transmitter &amp;t, SoundController &amp;sCtrl)</code>	46
5.11.3	Member Function Documentation	47
5.11.3.1	<code>button_pressed()</code>	47
5.11.3.2	<code>enable()</code>	47
5.11.3.3	<code>get_name()</code>	47
5.11.3.4	<code>main()</code>	48
5.11.3.5	<code>receive(Command c)</code>	48
5.11.3.6	<code>update_screen_game_parameters(bool alive)</code>	49
5.11.4	Member Data Documentation	50
5.11.4.1	<code>displayCtrl</code>	50
5.11.4.2	<code>enabled</code>	50
5.11.4.3	<code>game_timer</code>	50
5.11.4.4	<code>gameParameters</code>	50
5.11.4.5	<code>hit</code>	50



5.11.4.6	<a href="#">pressed</a>	50
5.11.4.7	<a href="#">soundCtrl</a>	50
5.11.4.8	<a href="#">transmitter</a>	51
5.12	<a href="#">Sound Struct Reference</a>	51
5.12.1	<a href="#">Detailed Description</a>	51
5.12.2	<a href="#">Member Data Documentation</a>	51
5.12.2.1	<a href="#">duration</a>	51
5.12.2.2	<a href="#">frequency</a>	51
5.13	<a href="#">SoundController Class Reference</a>	52
5.13.1	<a href="#">Detailed Description</a>	53
5.13.2	<a href="#">Constructor &amp; Destructor Documentation</a>	53
5.13.2.1	<a href="#">SoundController(hwlib::pin_out &amp;lsp)</a>	53
5.13.3	<a href="#">Member Function Documentation</a>	53
5.13.3.1	<a href="#">main()</a>	53
5.13.3.2	<a href="#">play(Sound s)</a>	53
5.13.3.3	<a href="#">play_shoot()</a>	54
5.13.4	<a href="#">Member Data Documentation</a>	54
5.13.4.1	<a href="#">lsp</a>	54
5.13.4.2	<a href="#">play_sound</a>	54
5.13.4.3	<a href="#">sounds</a>	54
5.14	<a href="#">Transmitter Class Reference</a>	55
5.14.1	<a href="#">Detailed Description</a>	56
5.14.2	<a href="#">Constructor &amp; Destructor Documentation</a>	56
5.14.2.1	<a href="#">Transmitter(const char *name, hwlib::target::d2_36kHz &amp;ir)</a>	56
5.14.3	<a href="#">Member Function Documentation</a>	56
5.14.3.1	<a href="#">main()</a>	56
5.14.3.2	<a href="#">send(short bits)</a>	56
5.14.4	<a href="#">Member Data Documentation</a>	57
5.14.4.1	<a href="#">command_bits</a>	57
5.14.4.2	<a href="#">command_received</a>	57
5.14.4.3	<a href="#">ir</a>	57

<b>6 File Documentation</b>	<b>59</b>
6.1 src/entities/gameParameters.cpp File Reference	59
6.2 src/entities/gameParameters.h File Reference	59
6.3 src/main.cpp File Reference	60
6.3.1 Enumeration Type Documentation	61
6.3.1.1 States	61
6.3.2 Function Documentation	61
6.3.2.1 main()	61
6.3.3 Variable Documentation	61
6.3.3.1 current_state	61
6.4 src/tasks/buttonController.cpp File Reference	62
6.5 src/tasks/buttonController.h File Reference	62
6.6 src/tasks/command.cpp File Reference	63
6.7 src/tasks/command.h File Reference	63
6.8 src/tasks/controller.h File Reference	64
6.9 src/tasks/displayController.cpp File Reference	65
6.10 src/tasks/displayController.h File Reference	66
6.11 src/tasks/initGameController.cpp File Reference	67
6.12 src/tasks/initGameController.h File Reference	67
6.13 src/tasks/receiver.cpp File Reference	68
6.14 src/tasks/receiver.h File Reference	69
6.15 src/tasks/registerController.cpp File Reference	70
6.16 src/tasks/registerController.h File Reference	71
6.17 src/tasks/runGameController.cpp File Reference	72
6.18 src/tasks/runGameController.h File Reference	72
6.19 src/tasks/soundController.cpp File Reference	73
6.20 src/tasks/soundController.h File Reference	74
6.21 src/tasks/transmitter.cpp File Reference	75
6.22 src/tasks/transmitter.h File Reference	76
<b>Index</b>	<b>77</b>

# 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:

Command . . . . .	11
Controller . . . . .	17
InitGameController . . . . .	24
RegisterController . . . . .	40
RunGameController . . . . .	45
GameParameters . . . . .	22
receive_shot . . . . .	34
Sound . . . . .	51
task	
ButtonController . . . . .	9
DisplayController . . . . .	20
InitGameController . . . . .	24
Main . . . . .	30
Receiver . . . . .	35
RegisterController . . . . .	40
RunGameController . . . . .	45
SoundController . . . . .	52
Transmitter . . . . .	55



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">ButtonController</a>	Can be interpreted as playing state and will handle each event during his state . . . . .	??
<a href="#">Command</a>	<a href="#">Command</a> 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 . . . . .	??
<a href="#">Controller</a>	Will be implented by each state . . . . .	??
<a href="#">DisplayController</a>	Will be used as communcation controller between oled boundary and other game state controllers . . . . .	??
<a href="#">GameParameters</a>	<a href="#">GameParameters</a> entity object will contain the player data . . . . .	??
<a href="#">InitGameController</a>	Can be interpetted as playing state and will handle each event during his state . . . . .	??
<a href="#">Main</a>	This class will resume and suspend running tasks based on it's current state . . . . .	??
<a href="#">receive_shot</a>	Receive_shot structure. Will be used for class <a href="#">GameParameters</a> . Each structure contains the player id and weapon id from the received hit . . . . .	??
<a href="#">Receiver</a>	Class that handles receiving IR handling . . . . .	??
<a href="#">RegisterController</a>	Can be interpret as register state and will handle each event during his state . . . . .	??
<a href="#">RunGameController</a>	Can be interpetted as playing state and will handle each event during his state . . . . .	??
<a href="#">Sound</a>	Simple struct to maintain frequency and duration data . . . . .	??
<a href="#">SoundController</a>	Simple task that can play a predefined sound . . . . .	??
<a href="#">Transmitter</a>	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 . . . . .	??





## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

src/main.cpp	??
src/entities/gameParameters.cpp	??
src/entities/gameParameters.h	??
src/tasks/buttonController.cpp	??
src/tasks/buttonController.h	??
src/tasks/command.cpp	??
src/tasks/command.h	??
src/tasks/controller.h	??
src/tasks/displayController.cpp	??
src/tasks/displayController.h	??
src/tasks/initGameController.cpp	??
src/tasks/initGameController.h	??
src/tasks/receiver.cpp	??
src/tasks/receiver.h	??
src/tasks/registerController.cpp	??
src/tasks/registerController.h	??
src/tasks/runGameController.cpp	??
src/tasks/runGameController.h	??
src/tasks/soundController.cpp	??
src/tasks/soundController.h	??
src/tasks/transmitter.cpp	??
src/tasks/transmitter.h	??



## Chapter 5

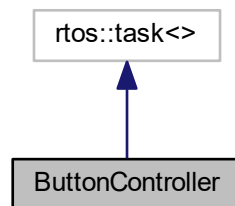
# Class Documentation

### 5.1 ButtonController Class Reference

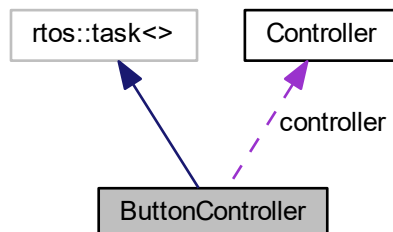
can be interpreted as playing state and will handle each event during his state.

```
#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* )

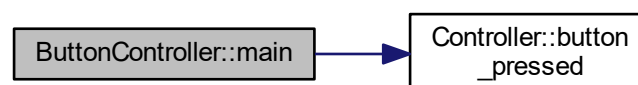
Constructor [ButtonController](#). /param ctrl reference for using controller interface /param gnd reference pin requires output pin /param vlt reference pin requires output pin /param b reference pin requires input pin

### 5.1.3 Member Function Documentation

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

RTOS task function

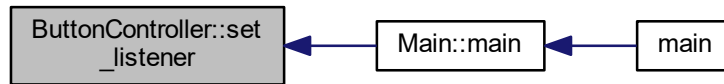
Here is the call graph for this function:



### 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/V2THDE%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:



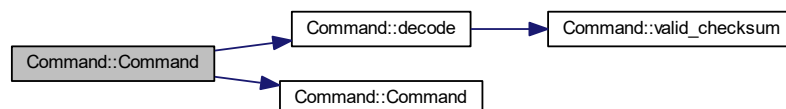
### 5.2.2.2 `Command::Command ( short bits ) [inline]`

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

#### Parameters

<i>bits</i>	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

<i>sender</i>	the sender part of the command
<i>data</i>	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

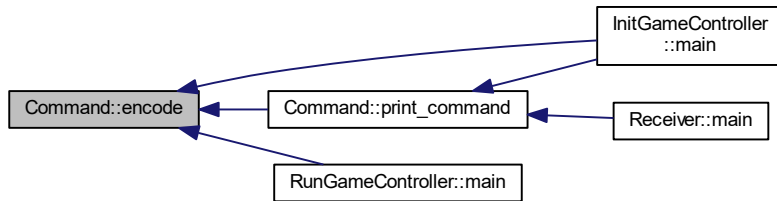
Decodes the int from the data into binary value and add it next to the short sender bits

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>



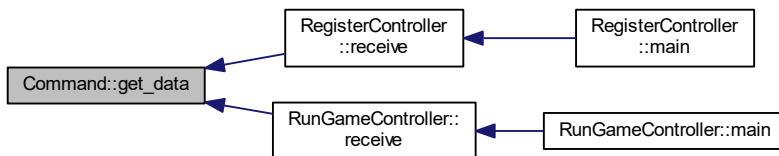
Here is the caller graph for this function:



### 5.2.3.3 `int Command::get_data ( )`

returns the data value

Here is the caller graph for this function:



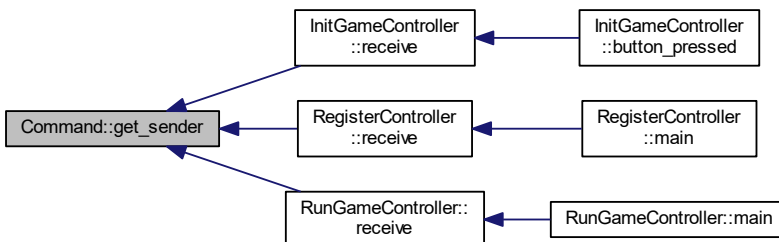
### 5.2.3.4 `bool Command::get_error ( )`

returns error value

### 5.2.3.5 `int Command::get_sender ( )`

returns the sender value

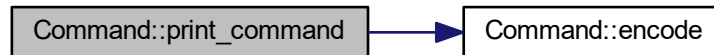
Here is the caller graph for this function:



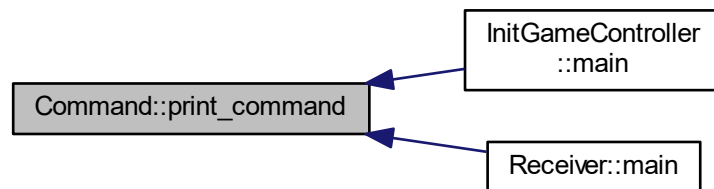
### 5.2.3.6 void Command::print\_command ( )

/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 )

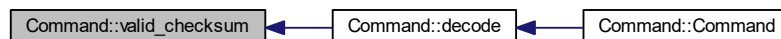
sets sender value

### 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:

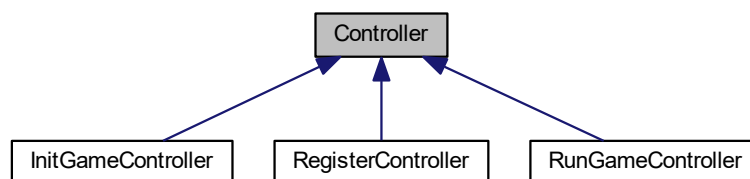
- [src/tasks/command.h](#)
- [src/tasks/command.cpp](#)

## 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

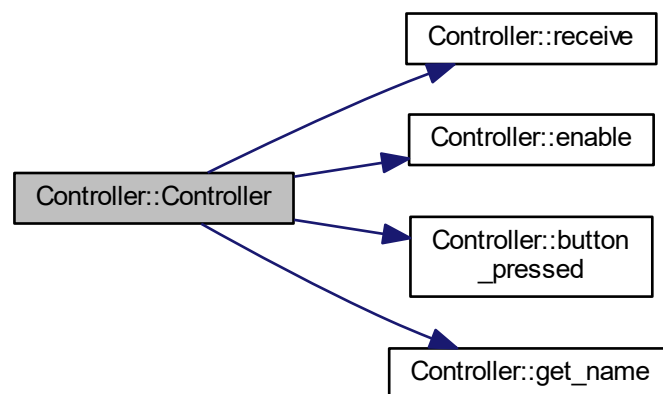
will be implented by each state

### 5.3.2 Constructor & Destructor Documentation

#### 5.3.2.1 `Controller::Controller ( )` `[inline]`

[Controller](#) Constructor

Here is the call graph for this function:



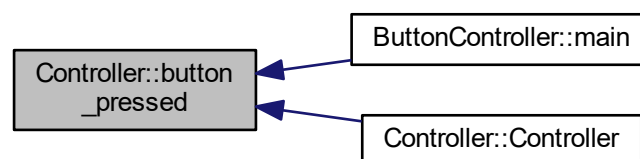
### 5.3.3 Member Function Documentation

#### 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:

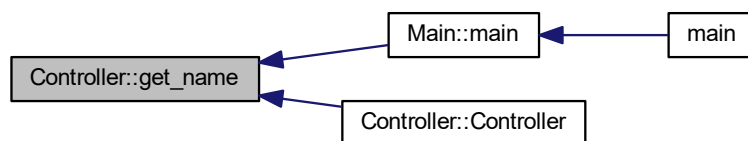


#### 5.3.3.3 `virtual const char* Controller::get_name ( ) [pure virtual]`

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:

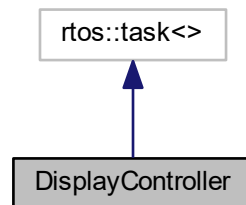
- `src/tasks/controller.h`

## 5.4 DisplayController Class Reference

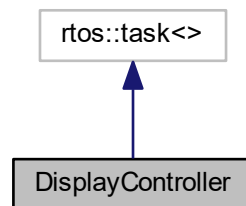
will be used as communication controller between oled boundary and other game state controllers

```
#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 communication 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 )

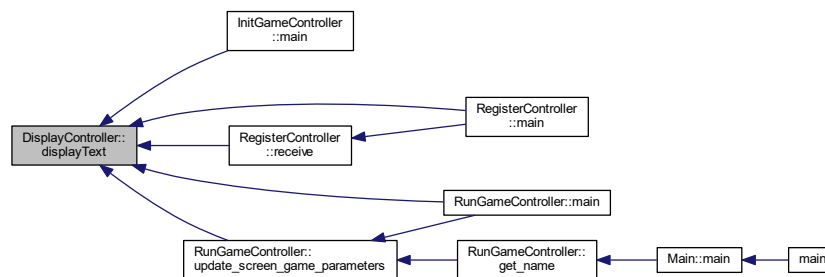
[DisplayController](#) Constructor /param hwlib::glcd\_oled\_buffered &c will be the oled boundary object for writing on screen

### 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:



#### 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:

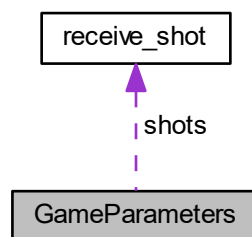
- [src/tasks/displayController.h](#)
- [src/tasks/displayController.cpp](#)

## 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 ( )`

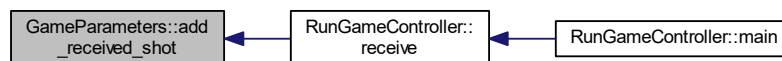
GameParameter Constructor

### 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:



### 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:

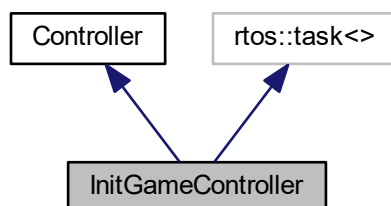
- [src/entities/gameParameters.h](#)
- [src/entities/gameParameters.cpp](#)

## 5.6 InitGameController Class Reference

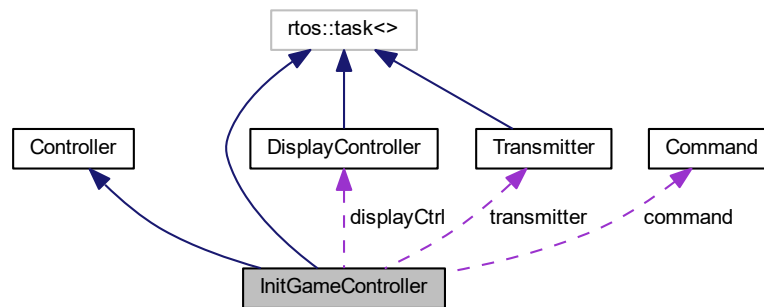
can be interpreted as playing state and will handle each event during his state.

```
#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

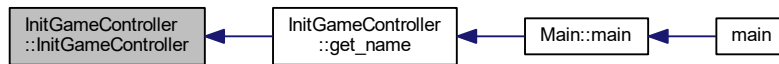
can be interpreted as playing state and will handle each event during his state.

## 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:



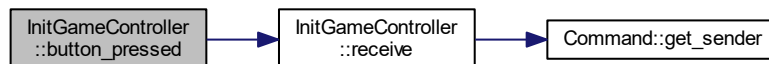
## 5.6.3 Member Function Documentation

### 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:

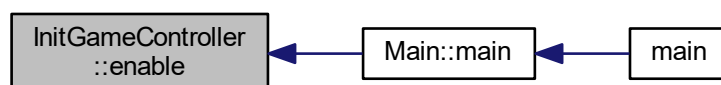


### 5.6.3.2 void InitGameController::enable ( ) [virtual]

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]`

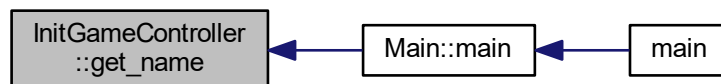
interface function for getting controller name

Implements [Controller](#).

Here is the call graph for this function:



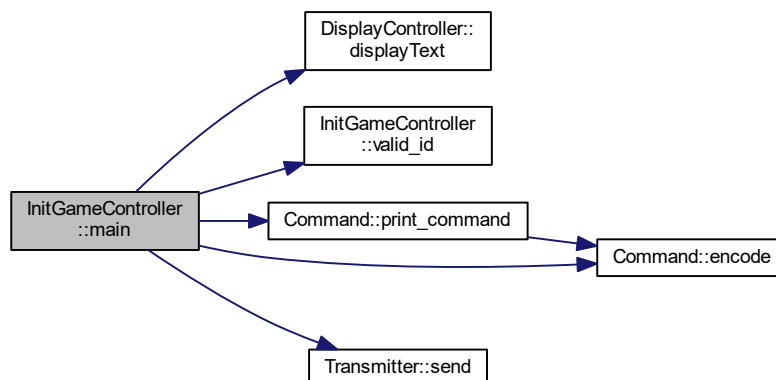
Here is the caller graph for this function:



### 5.6.3.4 `void InitGameController::main ( ) [private]`

RTOS main task function

Here is the call graph for this function:

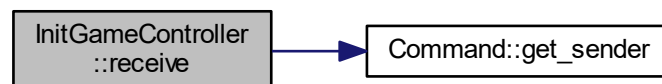


#### 5.6.3.5 void InitGameController::receive ( Command c ) [virtual]

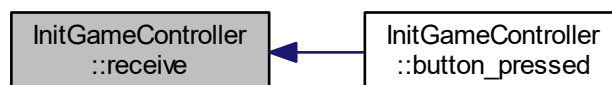
interface function for receiving command /param Command c

Implements [Controller](#).

Here is the call graph for this function:



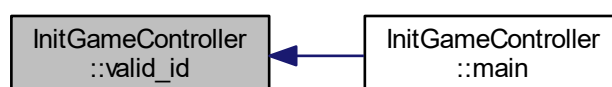
Here is the caller graph for this function:



#### 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 for 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 activate 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

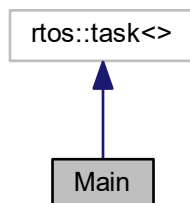
The documentation for this class was generated from the following files:

- [src/tasks/initGameController.h](#)
- [src/tasks/initGameController.cpp](#)

## 5.7 Main Class Reference

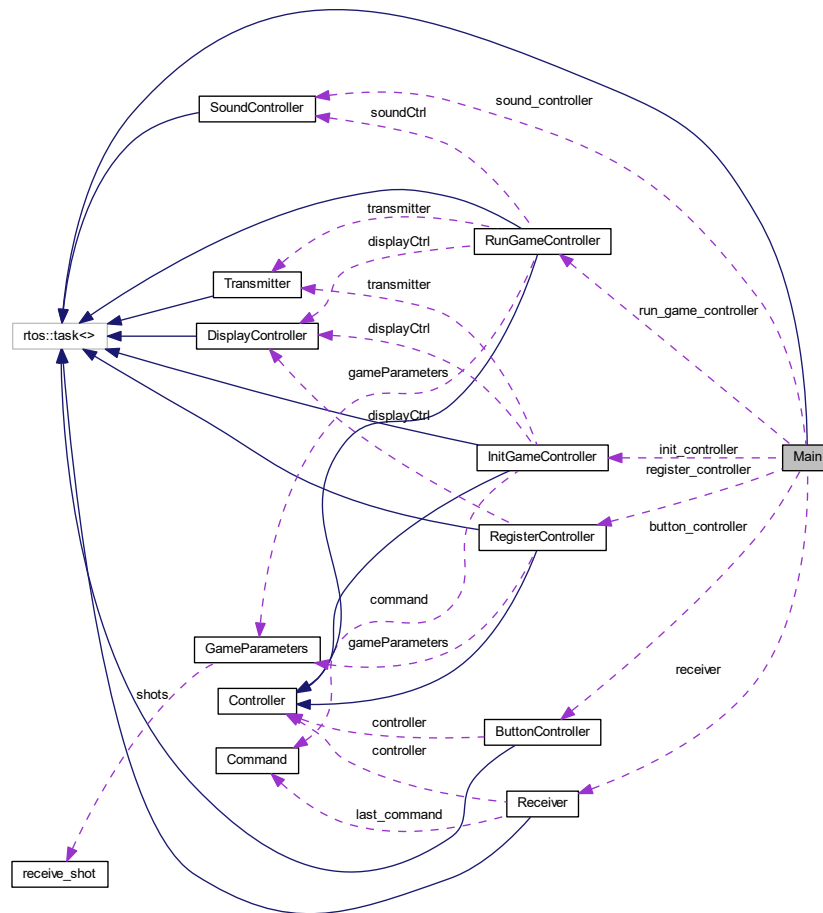
This class will resume and suspend running tasks based on it's current state.

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

<i>r</i>	a Reference to a existing instance of a <a href="#">Receiver</a> task
<i>i</i>	a Reference to a existing instance of a <a href="#">InitGameController</a> task
<i>reg</i>	a Reference to a existing instance of a <a href="#">RegisterController</a> task
<i>run</i>	a Reference to a existing instance of a <a href="#">RunGameFController</a> task

Here is the caller graph for this function:

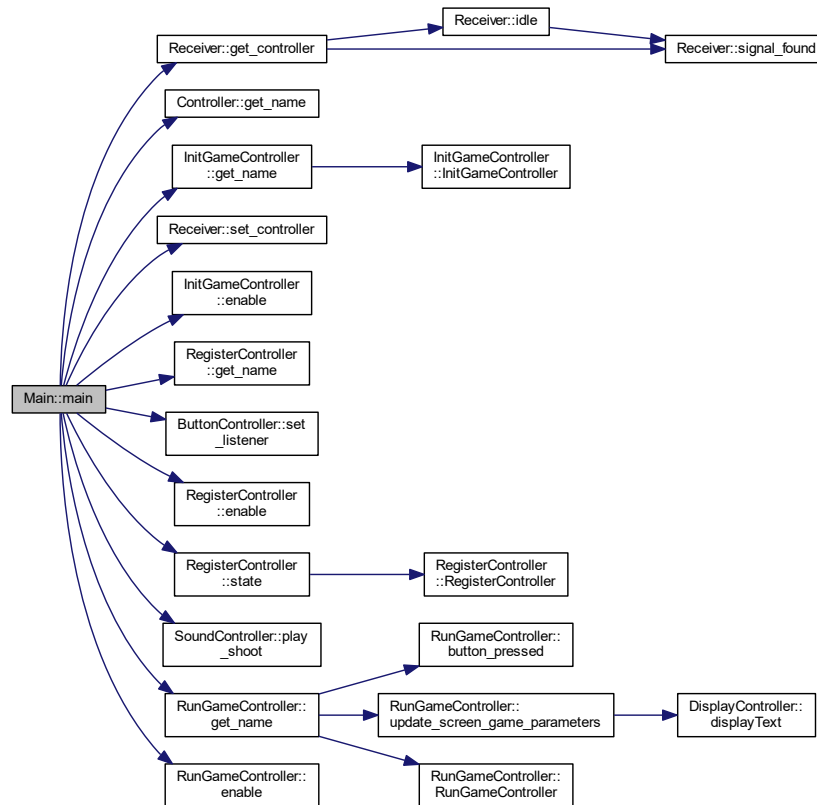


### 5.7.3 Member Function Documentation

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:



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:

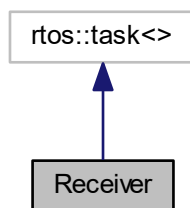
- [src/entities/gameParameters.h](#)

## 5.9 Receiver Class Reference

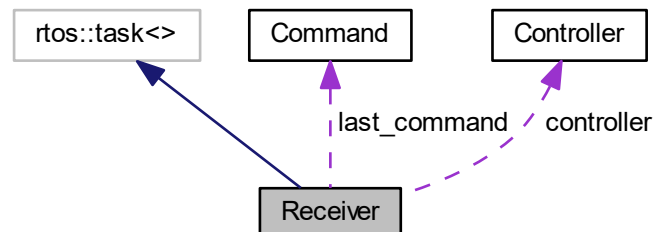
Class that handles receiving IR handling.

```
#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

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

## 5.9.3 Member Function Documentation

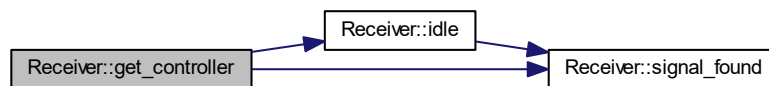
## 5.9.3.1 void Receiver::enable ( ) [inline]

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:

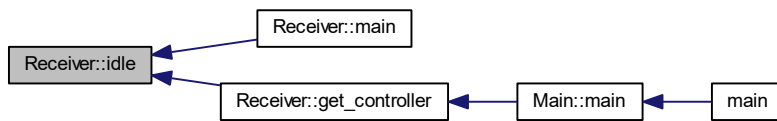


## 5.9.3.3 void Receiver::idle ( )

Here is the call graph for this function:



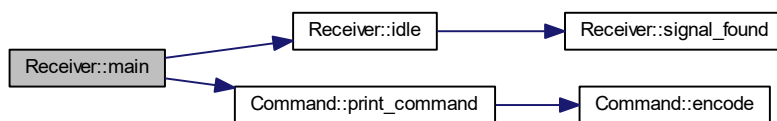
Here is the caller graph for this function:



#### 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 its `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

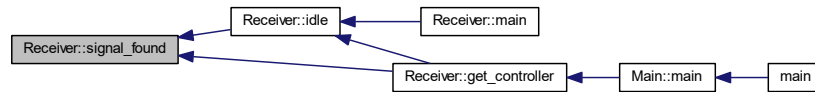
Here is the caller graph for this function:





## 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&amp; Receiver::signal [private]

Pin that receives the ir pulses.

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

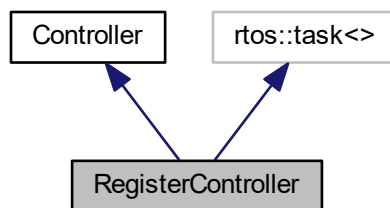
- [src/tasks/receiver.h](#)
- [src/tasks/receiver.cpp](#)

## 5.10 RegisterController Class Reference

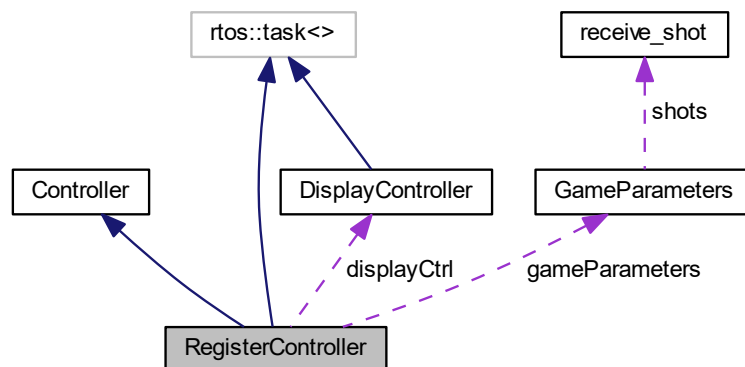
can be interpret as register state and will handle each event during his state.

```
#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:



### 5.10.3 Member Function Documentation

#### 5.10.3.1 void RegisterController::button\_pressed ( ) [virtual]

interface function to set button\_pressed flag

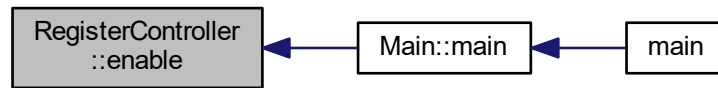
Implements [Controller](#).

### 5.10.3.2 void RegisterController::enable ( ) [virtual]

interface function to activate task function

Implements [Controller](#).

Here is the caller graph for this function:

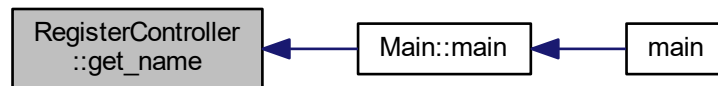


### 5.10.3.3 const char\* RegisterController::get\_name ( ) [inline],[virtual]

interface function to get name of the controller

Implements [Controller](#).

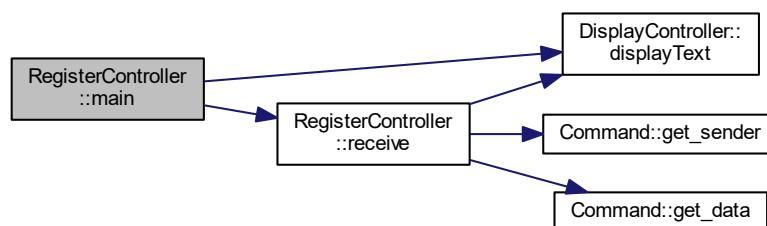
Here is the caller graph for this function:



### 5.10.3.4 void RegisterController::main ( ) [private]

RTOS main tas

Here is the call graph for this function:

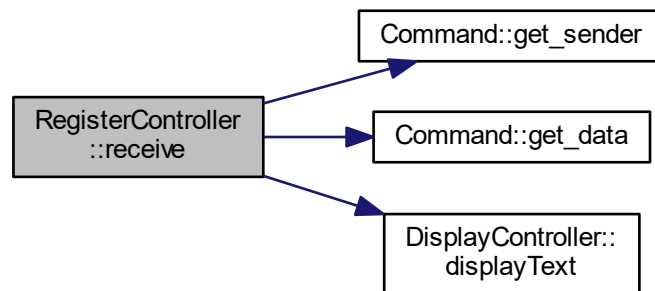


#### 5.10.3.5 void RegisterController::receive ( Command c ) [virtual]

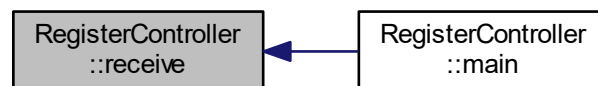
interface function for receiving data /param [Command](#) c to encode and decode data

Implements [Controller](#).

Here is the call graph for this function:



Here is the caller graph for this function:



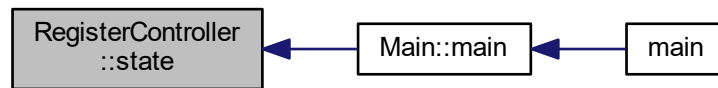
#### 5.10.3.6 int RegisterController::state ( ) [inline]

function for change stat

Here is the call graph for this function:



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:

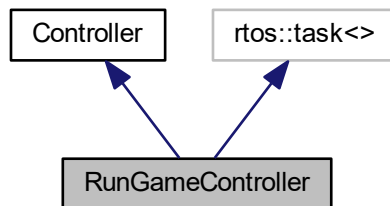
- [src/tasks/registerController.h](#)
- [src/tasks/registerController.cpp](#)

## 5.11 RunGameController Class Reference

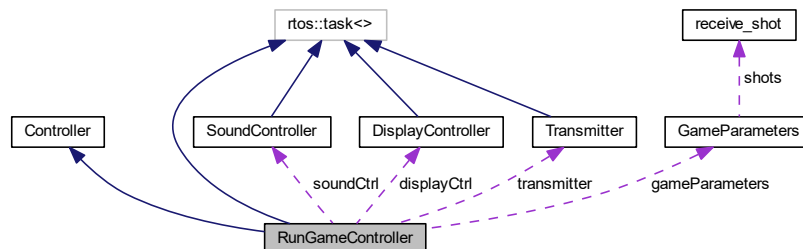
can be interpreted as playing state and will handle each event during his state.

```
#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, SoundController &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 storing 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 interpreted as playing state and will handle each event during his state.

### 5.11.2 Constructor & Destructor Documentation

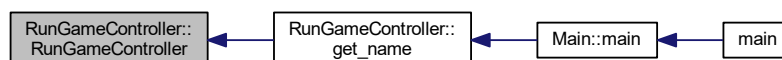
#### 5.11.2.1 [RunGameController::RunGameController](#) ( [GameParameters](#) & *gameParam*, [DisplayController](#) & *disCtrl*, [Transmitter](#) & *t*, [SoundController](#) & *sCtrl* )

[RunGameController](#) constructor.

#### Parameters

<i>&amp;gameParam</i>	to set or get player game settings.
<i>&amp;disCtrl</i>	to write text on screen.

Here is the caller graph for this function:





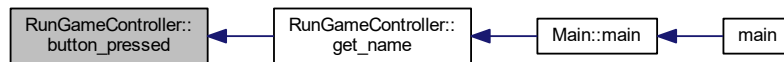
### 5.11.3 Member Function Documentation

#### 5.11.3.1 void RunGameController::button\_pressed ( ) [virtual]

button\_pressed interface function.

Implements [Controller](#).

Here is the caller graph for this function:

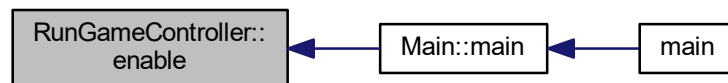


#### 5.11.3.2 void RunGameController::enable ( ) [virtual]

interface function for activating task.A

Implements [Controller](#).

Here is the caller graph for this function:

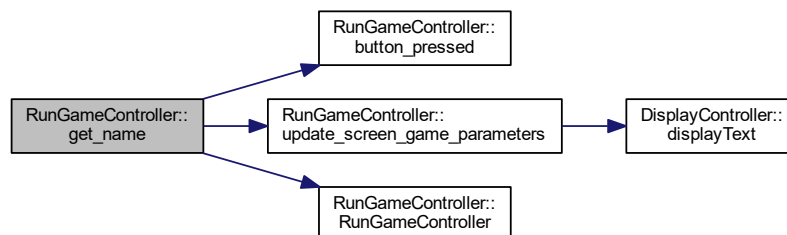


#### 5.11.3.3 const char\* RunGameController::get\_name ( ) [inline],[virtual]

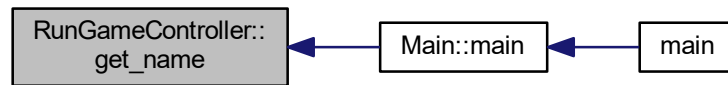
interface function for getting controller name.

Implements [Controller](#).

Here is the call graph for this function:



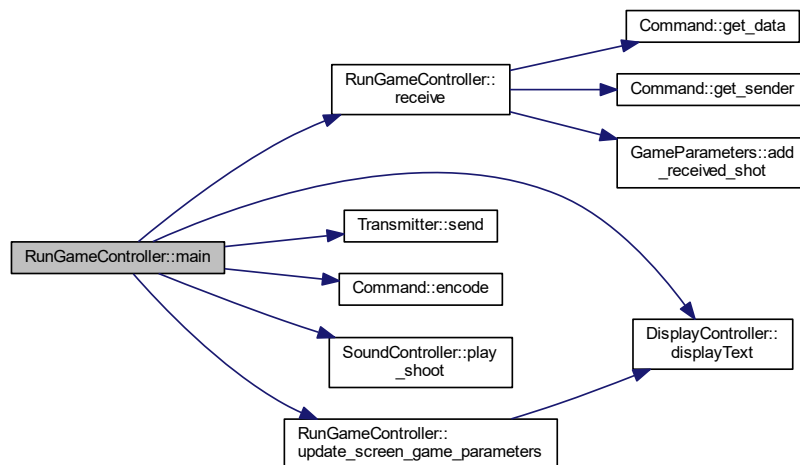
Here is the caller graph for this function:



#### 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]

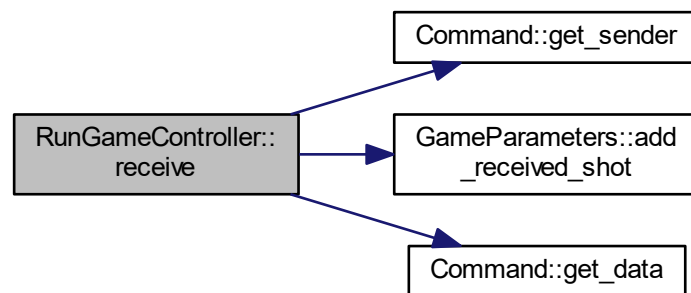
interface function for receiving a command

Parameters

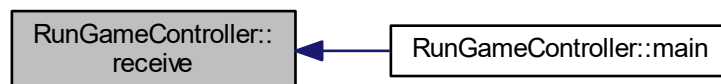
<a href="#">Command</a>	c for encode en decode data.
-------------------------	------------------------------

Implements [Controller](#).

Here is the call graph for this function:



Here is the caller graph for this function:



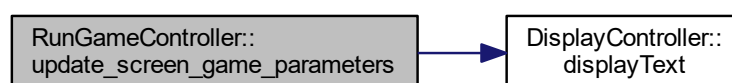
#### 5.11.3.6 void RunGameController::update\_screen\_game\_parameters ( bool *alive* )

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

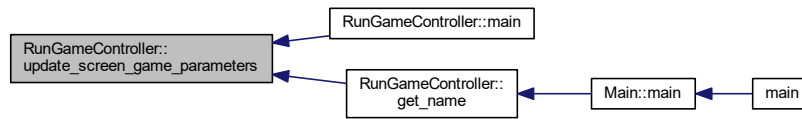
##### Parameters

<i>alive</i>	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 storing 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:

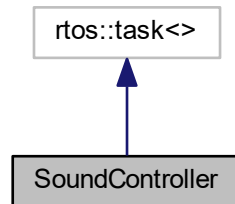
- src/tasks/[soundController.h](#)

## 5.13 SoundController Class Reference

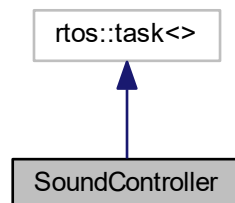
Simple task that can play a predefined sound.

```
#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

<i>lsp</i>	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
---	----------------------------------------------------

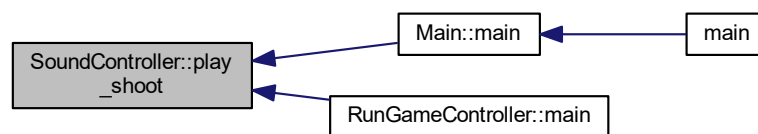
Here is the caller graph for this function:



### 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](#)

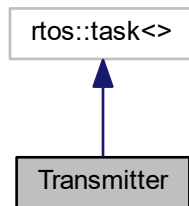


## 5.14 Transmitter Class Reference

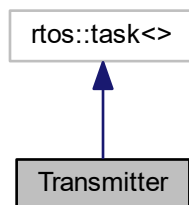
Tranmitter class used to send information over IR A rtos task that gives the user the ability of sending a SHORT through ir in binary form.

```
#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

Transmitter 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

<i>name</i>	Name of the task
<i>ir</i>	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

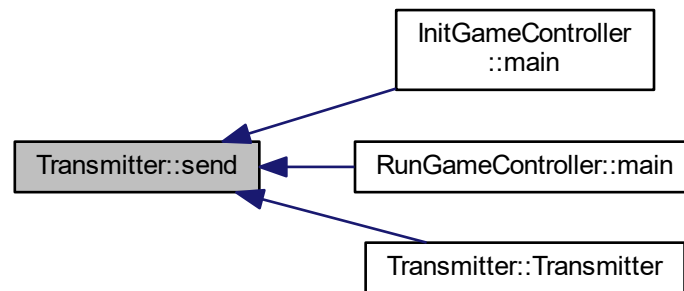
The command is sent twice in a delay of 3ms

The starbit is a simple 1

## Parameters

<i>bits</i>	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](#)



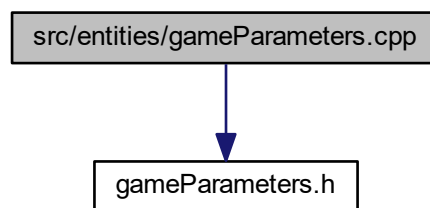
## Chapter 6

# File Documentation

### 6.1 src/entities/gameParameters.cpp File Reference

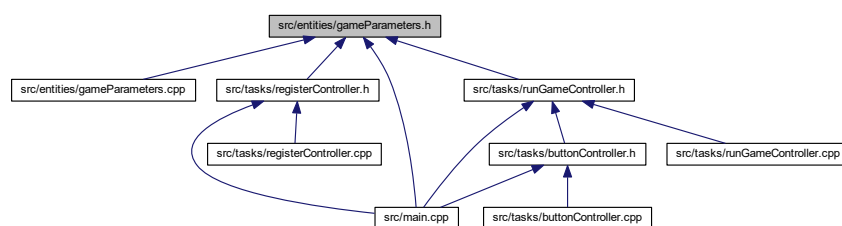
```
#include "gameParameters.h"
```

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:



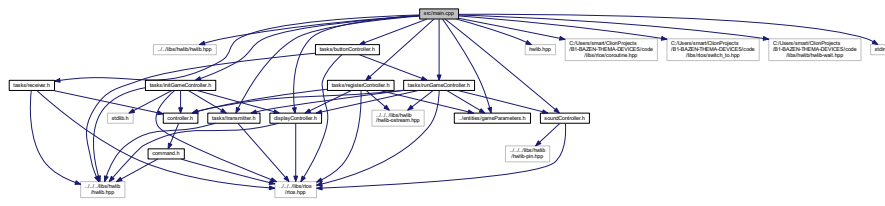
## Classes

- 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:



## 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**

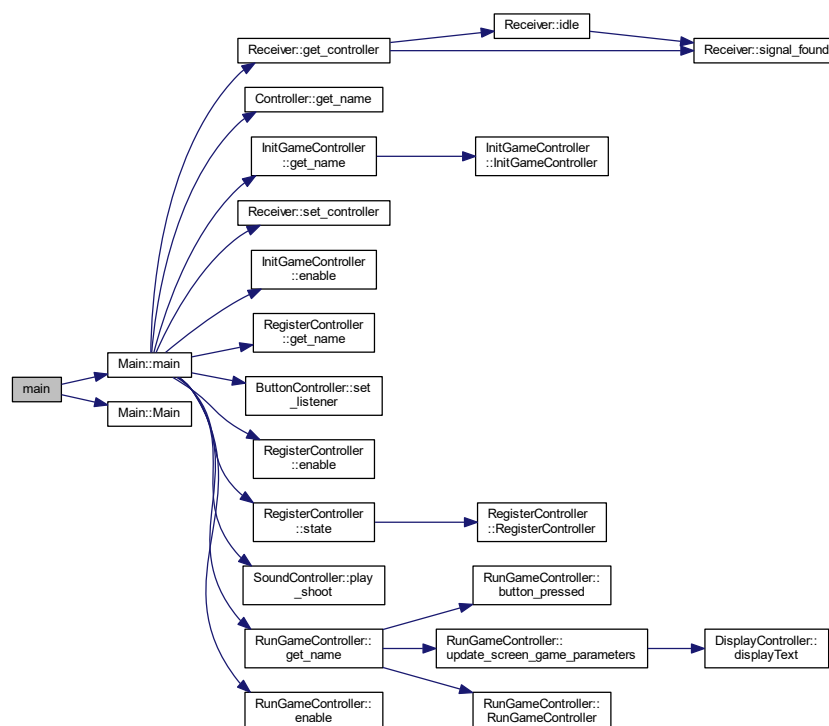
**RUNNING**

**GAME\_END**

## 6.3.2 Function Documentation

### 6.3.2.1 int main ( )

Here is the call graph for this function:



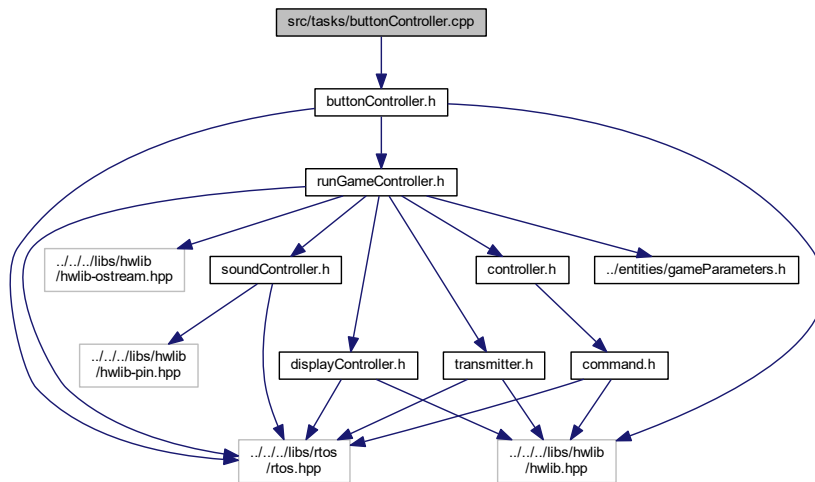
## 6.3.3 Variable Documentation

### 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:



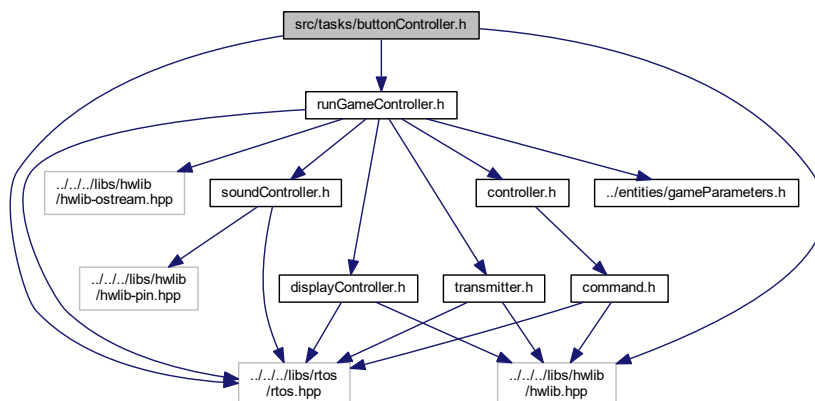
## 6.5 src/tasks/buttonController.h File Reference

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

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

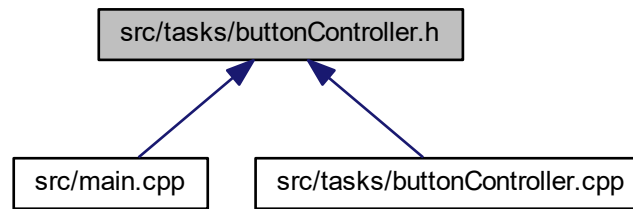
```
#include "runGameController.h"
```

Include dependency graph for buttonController.h:





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



## Classes

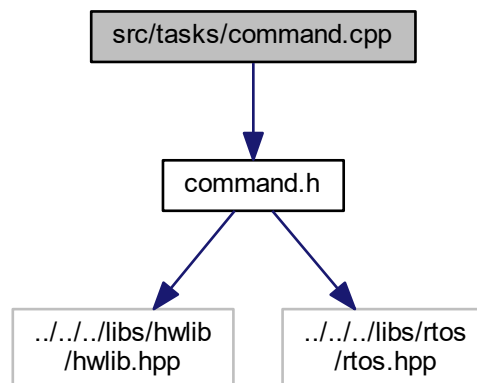
- class [ButtonController](#)

*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"
```

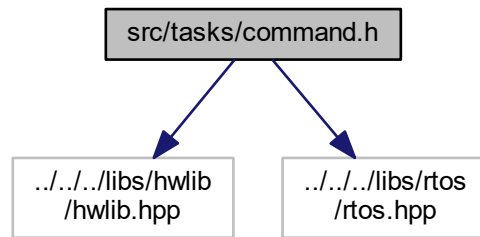
Include dependency graph for `command.cpp`:



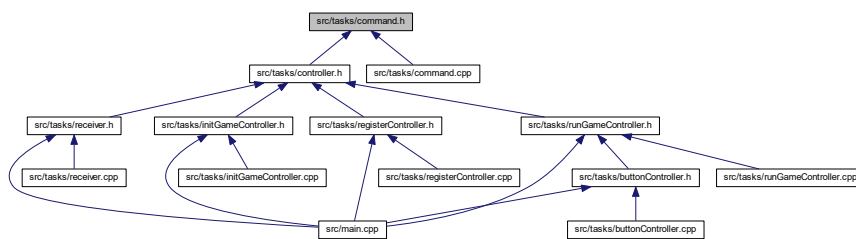
## 6.7 src/tasks/command.h File Reference

```
#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:



## Classes

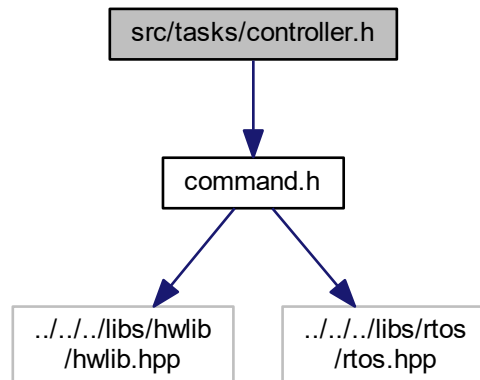
- class [Command](#)

*[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.*

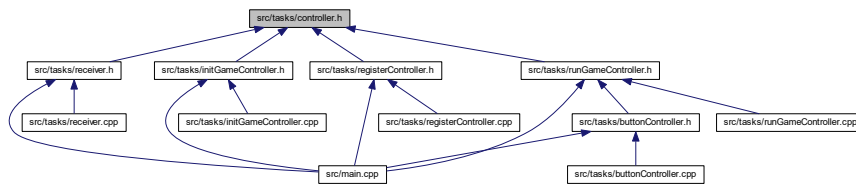
## 6.8 src/tasks/controller.h File Reference

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

Include dependency graph for controller.h:



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



## Classes

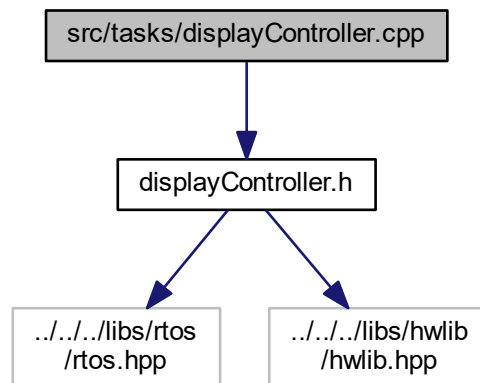
- class `Controller`

*will be implemented by each state*

## 6.9 src/tasks/displayController.cpp File Reference

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

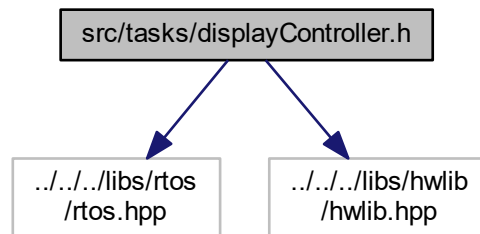
Include dependency graph for displayController.cpp:



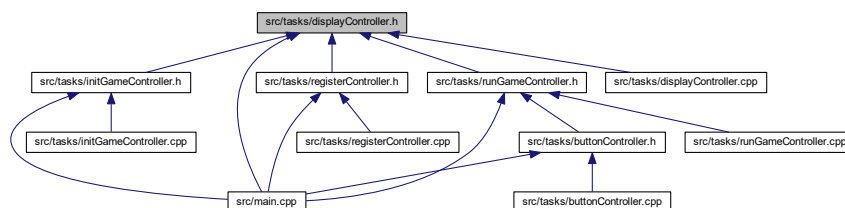
## 6.10 src/tasks/displayController.h File Reference

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

Include dependency graph for displayController.h:



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



## Classes

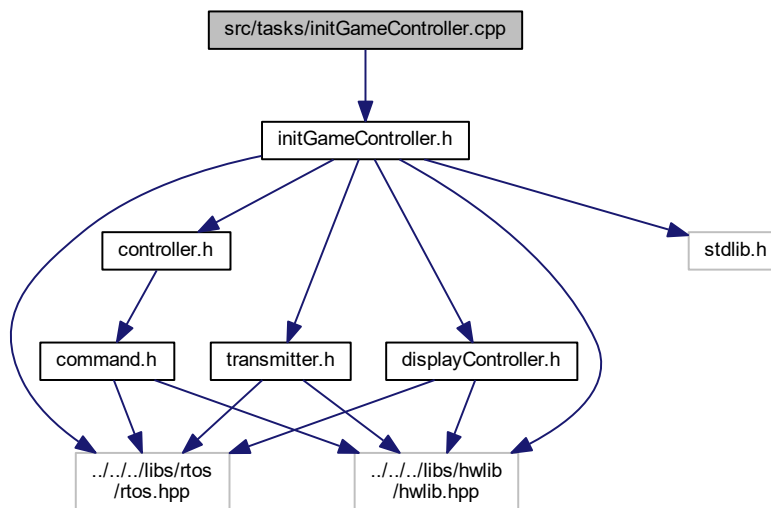
- class [DisplayController](#)

*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"
```

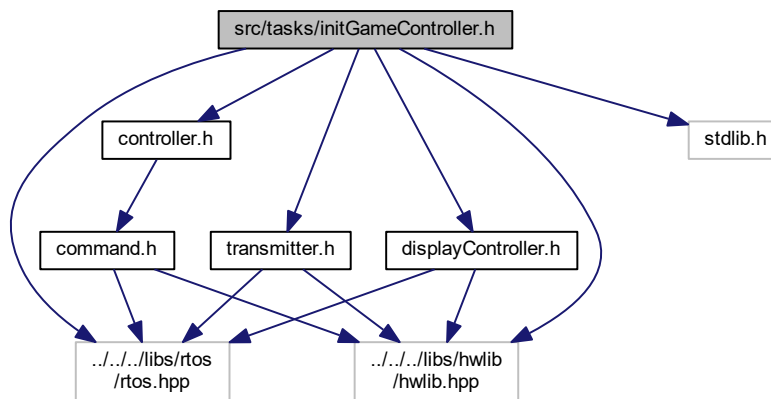
Include dependency graph for initGameController.cpp:



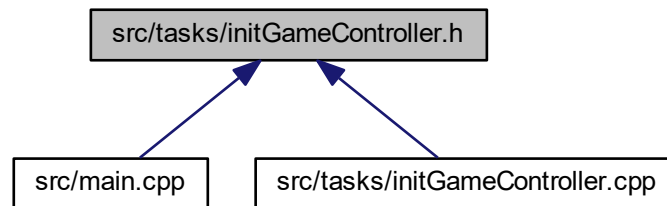
## 6.12 src/tasks/initGameController.h File Reference

```
#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:



## Classes

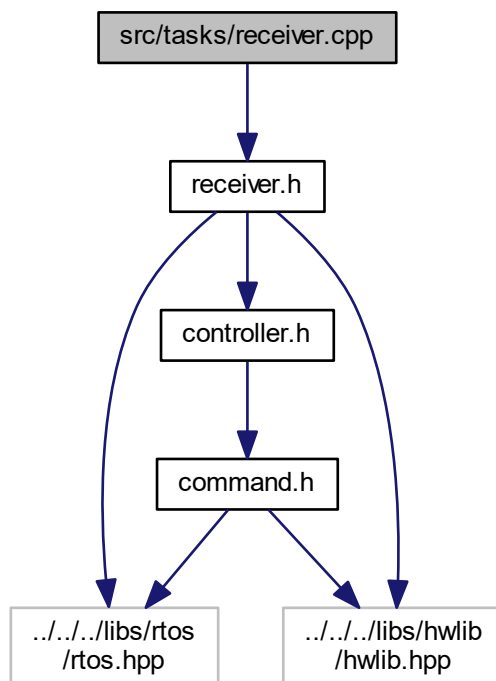
- class `InitGameController`

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

## 6.13 src/tasks/receiver.cpp File Reference

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

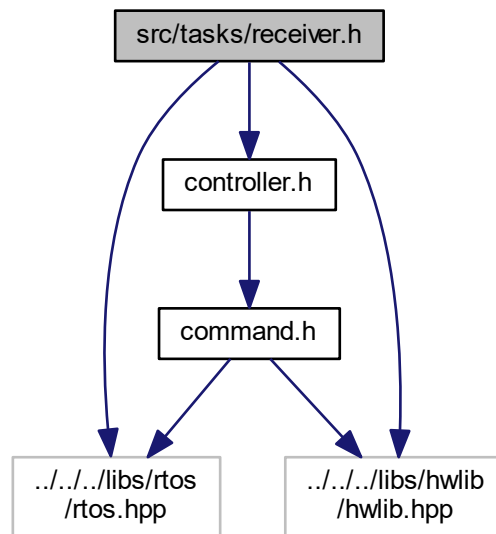
Include dependency graph for receiver.cpp:



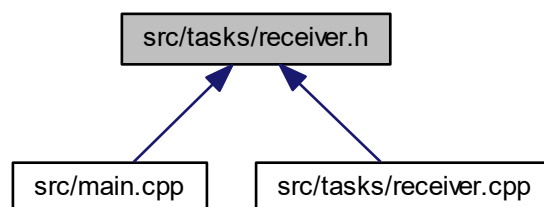
## 6.14 src/tasks/receiver.h File Reference

```
#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

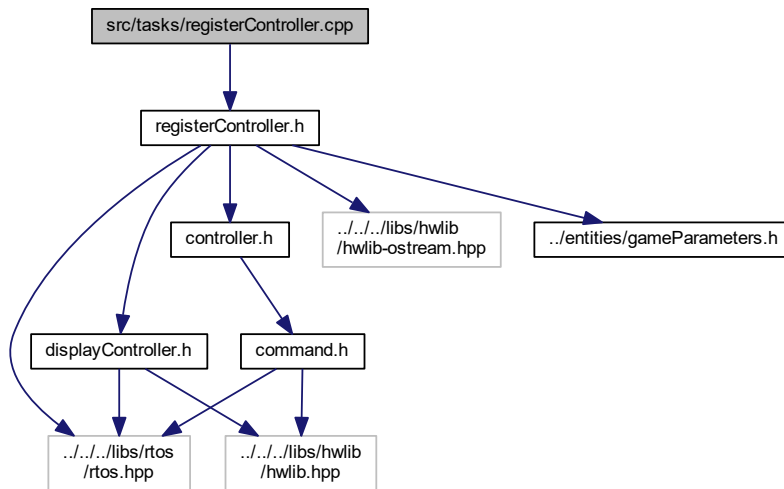
- class [Receiver](#)  
*Class that handles receiving IR handling.*

## 6.15 src/tasks/registerController.cpp File Reference

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



Include dependency graph for registerController.cpp:



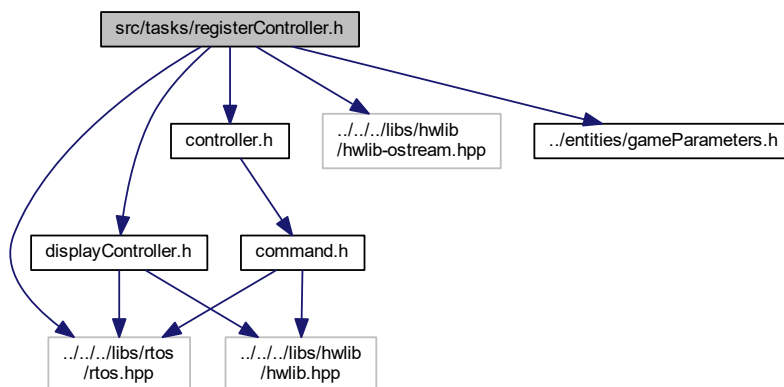
## 6.16 src/tasks/registerController.h File Reference

```

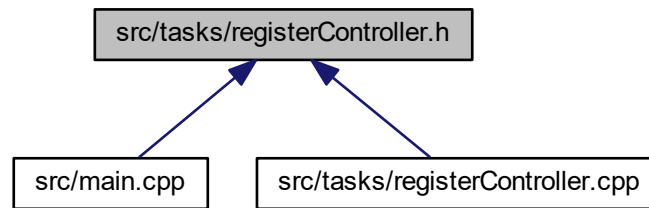
#include "../../../../libs/rtos/rtos.hpp"
#include "controller.h"
#include "../../../../libs/hwlib/hwlib-ostream.hpp"
#include "displayController.h"
#include "../entities/gameParameters.h"

```

Include dependency graph for registerController.h:



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



## Classes

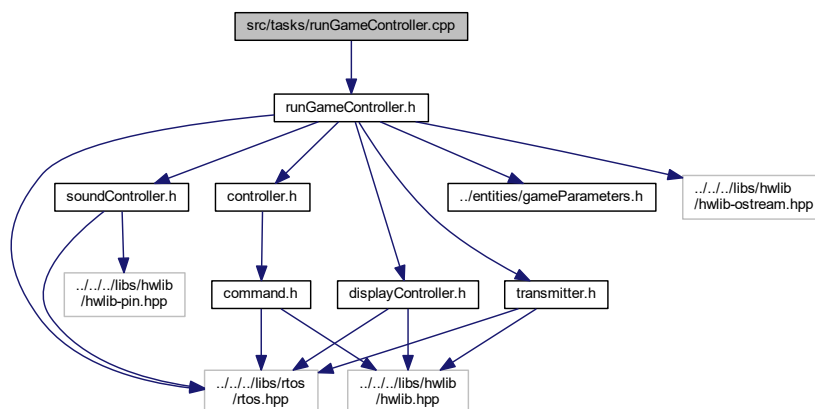
- class [RegisterController](#)

*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`:

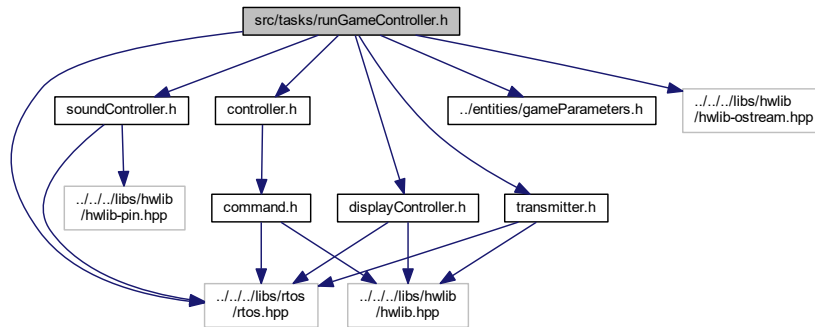


## 6.18 src/tasks/runGameController.h File Reference

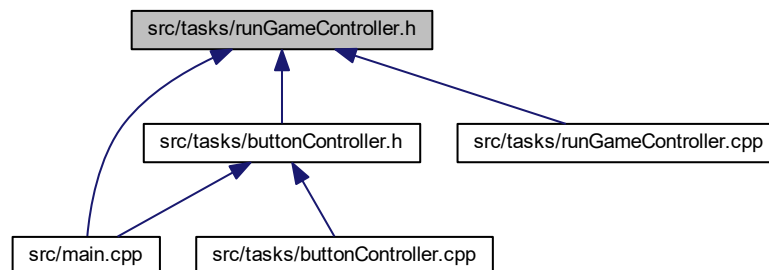
```
#include "../libs/rtos/rtos.hpp"
```

```
#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:



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



## Classes

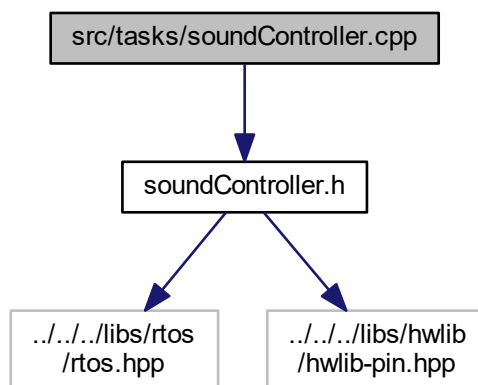
- class [RunGameController](#)

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

## 6.19 src/tasks/soundController.cpp File Reference

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

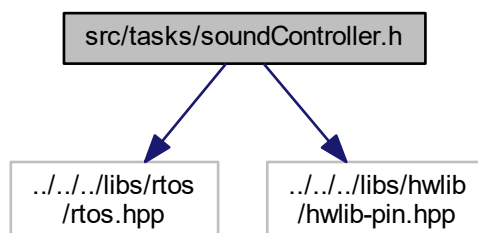
Include dependency graph for soundController.cpp:



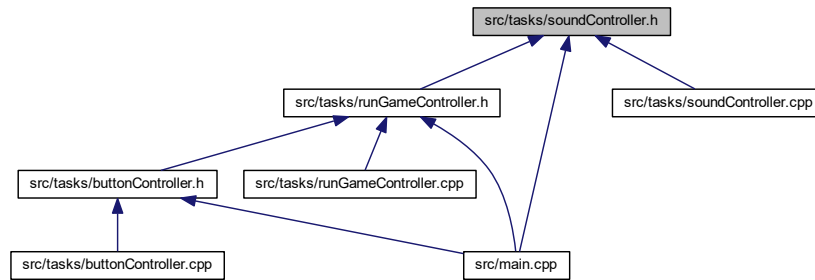
## 6.20 src/tasks/soundController.h File Reference

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

Include dependency graph for soundController.h:



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



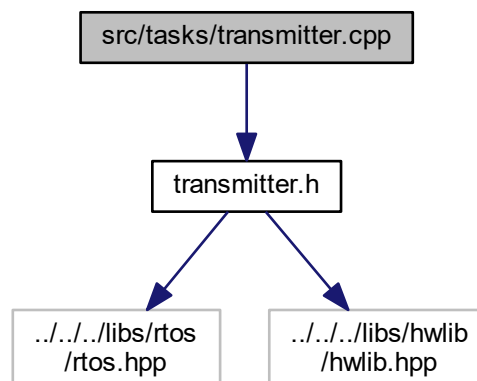
## Classes

- 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"
```

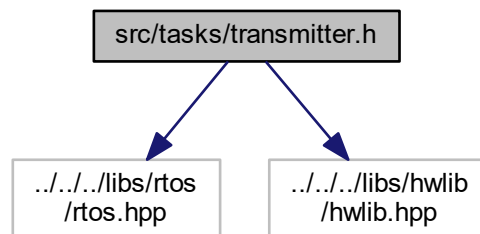
Include dependency graph for transmitter.cpp:



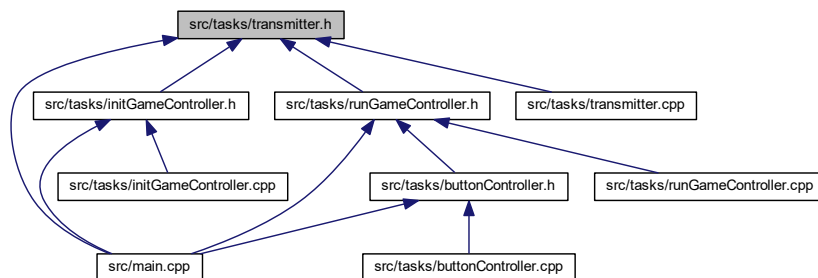
## 6.22 src/tasks/transmitter.h File Reference

```
#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:



## 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

- add\_received\_shot
  - GameParameters, [23](#)
- amount\_bits\_found
  - Receiver, [39](#)
- bits
  - Receiver, [39](#)
- buffer
  - DisplayController, [21](#)
- button
  - ButtonController, [11](#)
- button\_controller
  - Main, [33](#)
- button\_pressed
  - Controller, [18](#)
  - InitGameController, [26](#)
  - RegisterController, [41](#)
  - RunGameController, [47](#)
- ButtonController, [9](#)
  - button, [11](#)
  - ButtonController, [10](#)
  - clock, [11](#)
  - controller, [11](#)
  - ground, [11](#)
  - main, [10](#)
  - set\_listener, [10](#)
  - voltage, [11](#)
- clearFlag
  - DisplayController, [21](#)
- clock
  - ButtonController, [11](#)
- Command, [11](#)
  - Command, [13](#)
  - data, [17](#)
  - decode, [14](#)
  - encode, [14](#)
  - error, [17](#)
  - get\_data, [15](#)
  - get\_error, [15](#)
  - get\_sender, [15](#)
  - print\_command, [15](#)
  - sender, [17](#)
  - set\_data, [16](#)
  - set\_sender, [16](#)
  - valid\_checksum, [16](#)
- command
  - InitGameController, [29](#)
- command\_available
  - InitGameController, [29](#)
- command\_bits
  - Transmitter, [57](#)
- command\_full
  - InitGameController, [29](#)
- command\_processed
  - RegisterController, [44](#)
- command\_received
  - Transmitter, [57](#)
- Controller, [17](#)
  - button\_pressed, [18](#)
  - Controller, [18](#)
  - enable, [18](#)
  - get\_name, [19](#)
  - receive, [19](#)
- controller
  - ButtonController, [11](#)
  - Receiver, [39](#)
- current\_state
  - main.cpp, [61](#)
- custom\_command
  - InitGameController, [29](#)
- data
  - Command, [17](#)
- decode
  - Command, [14](#)
- DisplayController, [20](#)
  - buffer, [21](#)
  - clearFlag, [21](#)
  - DisplayController, [21](#)
  - displayText, [21](#)
  - flushFlag, [21](#)
  - main, [21](#)
  - oled, [22](#)
  - timer\_screen, [22](#)
- displayCtrl
  - InitGameController, [29](#)
  - RegisterController, [44](#)
  - RunGameController, [50](#)
- displayText
  - DisplayController, [21](#)
- duration
  - Sound, [51](#)
- enable
  - Controller, [18](#)
  - InitGameController, [26](#)
  - Receiver, [37](#)
  - RegisterController, [41](#)
  - RunGameController, [47](#)

- enabled
  - InitGameController, 29
  - Receiver, 39
  - RegisterController, 44
  - RunGameController, 50
- encode
  - Command, 14
- error
  - Command, 17
- flushFlag
  - DisplayController, 21
- frequency
  - Sound, 51
- GAME\_END
  - main.cpp, 61
- game\_time
  - GameParameters, 23
- game\_timer
  - RunGameController, 50
- GameParameters, 22
  - add\_received\_shot, 23
  - game\_time, 23
  - GameParameters, 23
  - health, 23
  - id, 24
  - shots, 24
  - shots\_taken, 24
  - weapon, 24
- gameParameters
  - RegisterController, 44
  - RunGameController, 50
- get\_controller
  - Receiver, 37
- get\_data
  - Command, 15
- get\_error
  - Command, 15
- get\_name
  - Controller, 19
  - InitGameController, 26
  - RegisterController, 42
  - RunGameController, 47
- get\_sender
  - Command, 15
- ground
  - ButtonController, 11
- health
  - GameParameters, 23
- hit
  - RunGameController, 50
- INIT
  - main.cpp, 61
- id
  - GameParameters, 24
- idle
  - Receiver, 37
- init\_controller
  - Main, 33
- InitGameController, 24
  - button\_pressed, 26
  - command, 29
  - command\_available, 29
  - command\_full, 29
  - custom\_command, 29
  - displayCtrl, 29
  - enable, 26
  - enabled, 29
  - get\_name, 26
  - InitGameController, 26
  - keypad, 29
  - main, 27
  - player\_id, 29
  - receive, 27
  - transmitter, 29
  - valid\_id, 28
  - weapon\_id, 29
- ir
  - Transmitter, 57
- keypad
  - InitGameController, 29
- last\_command
  - Receiver, 39
- lsp
  - SoundController, 54
- Main, 30
  - button\_controller, 33
  - init\_controller, 33
  - Main, 32
  - main, 32
  - receiver, 33
  - register\_controller, 34
  - run\_game\_controller, 34
  - sound\_controller, 34
- main
  - ButtonController, 10
  - DisplayController, 21
  - InitGameController, 27
  - Main, 32
  - main.cpp, 61
  - Receiver, 38
  - RegisterController, 42
  - RunGameController, 48
  - SoundController, 53
  - Transmitter, 56
- main.cpp
  - current\_state, 61
  - GAME\_END, 61
  - INIT, 61
  - main, 61
  - REGISTER, 61
  - RUNNING, 61



- States, 61
- max\_bits
  - Receiver, 39
- next\_state
  - RegisterController, 44
- oled
  - DisplayController, 22
- play
  - SoundController, 53
- play\_shoot
  - SoundController, 54
- play\_sound
  - SoundController, 54
- player\_id
  - InitGameController, 29
  - receive\_shot, 34
- pressed
  - RegisterController, 44
  - RunGameController, 50
- print\_command
  - Command, 15
- REGISTER
  - main.cpp, 61
- RUNNING
  - main.cpp, 61
- ready\_to\_receive
  - RegisterController, 44
- receive
  - Controller, 19
  - InitGameController, 27
  - RegisterController, 42
  - RunGameController, 48
- receive\_shot, 34
  - player\_id, 34
  - weapon\_id, 34
- Receiver, 35
  - amount\_bits\_found, 39
  - bits, 39
  - controller, 39
  - enable, 37
  - enabled, 39
  - get\_controller, 37
  - idle, 37
  - last\_command, 39
  - main, 38
  - max\_bits, 39
  - Receiver, 36
  - set\_controller, 38
  - signal, 39
  - signal\_found, 38
- receiver
  - Main, 33
- register\_controller
  - Main, 34
- RegisterController, 40
  - button\_pressed, 41
  - command\_processed, 44
  - displayCtrl, 44
  - enable, 41
  - enabled, 44
  - gameParameters, 44
  - get\_name, 42
  - main, 42
  - next\_state, 44
  - pressed, 44
  - ready\_to\_receive, 44
  - receive, 42
  - RegisterController, 41
  - state, 43
- run\_game\_controller
  - Main, 34
- RunGameController, 45
  - button\_pressed, 47
  - displayCtrl, 50
  - enable, 47
  - enabled, 50
  - game\_timer, 50
  - gameParameters, 50
  - get\_name, 47
  - hit, 50
  - main, 48
  - pressed, 50
  - receive, 48
  - RunGameController, 46
  - soundCtrl, 50
  - transmitter, 50
  - update\_screen\_game\_parameters, 49
- send
  - Transmitter, 56
- sender
  - Command, 17
- set\_controller
  - Receiver, 38
- set\_data
  - Command, 16
- set\_listener
  - ButtonController, 10
- set\_sender
  - Command, 16
- shots
  - GameParameters, 24
- shots\_taken
  - GameParameters, 24
- signal
  - Receiver, 39
- signal\_found
  - Receiver, 38
- Sound, 51
  - duration, 51
  - frequency, 51
- sound\_controller
  - Main, 34
- SoundController, 52

- lsp, 54
- main, 53
- play, 53
- play\_shoot, 54
- play\_sound, 54
- SoundController, 53
- sounds, 54
- soundCtrl
  - RunGameController, 50
- sounds
  - SoundController, 54
- src/entities/gameParameters.cpp, 59
- src/entities/gameParameters.h, 59
- src/main.cpp, 60
- src/tasks/buttonController.cpp, 62
- src/tasks/buttonController.h, 62
- src/tasks/command.cpp, 63
- src/tasks/command.h, 63
- src/tasks/controller.h, 64
- src/tasks/displayController.cpp, 65
- src/tasks/displayController.h, 66
- src/tasks/initGameController.cpp, 67
- src/tasks/initGameController.h, 67
- src/tasks/receiver.cpp, 68
- src/tasks/receiver.h, 69
- src/tasks/registerController.cpp, 70
- src/tasks/registerController.h, 71
- src/tasks/runGameController.cpp, 72
- src/tasks/runGameController.h, 72
- src/tasks/soundController.cpp, 73
- src/tasks/soundController.h, 74
- src/tasks/transmitter.cpp, 75
- src/tasks/transmitter.h, 76
- state
  - RegisterController, 43
- States
  - main.cpp, 61
- timer\_screen
  - DisplayController, 22
- Transmitter, 55
  - command\_bits, 57
  - command\_received, 57
  - ir, 57
  - main, 56
  - send, 56
  - Transmitter, 56
- transmitter
  - InitGameController, 29
  - RunGameController, 50
- update\_screen\_game\_parameters
  - RunGameController, 49
- valid\_checksum
  - Command, 16
- valid\_id
  - InitGameController, 28
- voltage
  - ButtonController, 11
- weapon
  - GameParameters, 24
- weapon\_id
  - InitGameController, 29
  - receive\_shot, 34