

Pibox

Generated by Doxygen 1.8.14

Contents

1	Data Structure Index	1
1.1	Data Structures	1
2	File Index	3
2.1	File List	3
3	Data Structure Documentation	5
3.1	BUFFERS Struct Reference	5
3.1.1	Detailed Description	5
3.1.2	Field Documentation	6
3.1.2.1	buff1_l	6
3.1.2.2	buff1_r	6
3.1.2.3	buff2_l	6
3.1.2.4	buff2_r	6
3.1.2.5	currentBuffer	6
3.1.2.6	flags	7
3.1.2.7	lengthBuffer	7
3.1.2.8	sampleReaded	7
3.2	fsm_ Struct Reference	7
3.2.1	Detailed Description	7
3.2.2	Field Documentation	8
3.2.2.1	current_state	8
3.2.2.2	tt	8
3.2.2.3	user_data	8

3.3	fsm_audio_controller Struct Reference	8
3.3.1	Detailed Description	9
3.3.2	Field Documentation	9
3.3.2.1	buffer	9
3.3.2.2	fsm	9
3.3.2.3	tipo_sistema	9
3.4	fsm_trans_ Struct Reference	9
3.4.1	Detailed Description	10
3.4.2	Field Documentation	10
3.4.2.1	dest_state	10
3.4.2.2	in	10
3.4.2.3	orig_state	10
3.4.2.4	out	10
3.5	ISR_Typ Struct Reference	11
3.5.1	Detailed Description	11
3.5.2	Field Documentation	11
3.5.2.1	callback	11
3.5.2.2	event	11
3.5.2.3	pin	11
3.6	list_files Struct Reference	12
3.6.1	Detailed Description	12
3.6.2	Field Documentation	12
3.6.2.1	current_file	12
3.6.2.2	name_file	12
3.6.2.3	num_files	13
3.6.2.4	select_file	13
3.7	TipoSistema Struct Reference	13
3.7.1	Detailed Description	13

4 File Documentation	15
4.1 dbcontroller.c File Reference	15
4.1.1 Macro Definition Documentation	16
4.1.1.1 SQL_QUERY_CHECK_DB	16
4.1.1.2 SQL_QUERY_CREATE_DB	16
4.1.1.3 SQL_QUERY_GET_NUMBER_OF_ELEMENTS	16
4.1.1.4 SQL_QUERY_INSERT_DB	16
4.1.1.5 SQL_QUERY_MAX_SIZE	16
4.1.1.6 SQL_QUERY_SELECT_DB	17
4.1.1.7 SQL_TABLE_NAME	17
4.1.2 Function Documentation	17
4.1.2.1 db_check()	17
4.1.2.2 db_close()	18
4.1.2.3 db_create_tables()	18
4.1.2.4 db_free_song_name()	19
4.1.2.5 db_get_song_name()	19
4.1.2.6 db_insert()	20
4.1.2.7 db_load()	21
4.2 dbcontroller.h File Reference	21
4.2.1 Enumeration Type Documentation	22
4.2.1.1 SQL_RESOLUTION	22
4.2.2 Function Documentation	22
4.2.2.1 db_check()	22
4.2.2.2 db_close()	23
4.2.2.3 db_get_song_name()	23
4.2.2.4 db_insert()	24
4.2.2.5 db_load()	25
4.3 defines.h File Reference	25
4.3.1 Macro Definition Documentation	25
4.3.1.1 bcm_spi	25

4.3.1.2	CANCION_ACABADA	26
4.3.1.3	CANCION_NOACABADA	26
4.3.1.4	PIN_PWM	26
4.3.1.5	use_bcm	26
4.4	fsm.c File Reference	26
4.4.1	Function Documentation	26
4.4.1.1	fsm_delete()	27
4.4.1.2	fsm_fire()	27
4.4.1.3	fsm_new()	27
4.5	fsm.h File Reference	27
4.5.1	Typedef Documentation	28
4.5.1.1	fsm_input_func_t	28
4.5.1.2	fsm_output_func_t	28
4.5.1.3	fsm_t	28
4.5.1.4	fsm_trans_t	29
4.5.2	Function Documentation	29
4.5.2.1	fsm_delete()	29
4.5.2.2	fsm_fire()	29
4.5.2.3	fsm_new()	29
4.6	fsm_rfid.c File Reference	30
4.6.1	Macro Definition Documentation	32
4.6.1.1	DB_NAME	32
4.6.1.2	PIN_A	32
4.6.1.3	PIN_B	32
4.6.1.4	PIN_C	32
4.6.2	Function Documentation	32
4.6.2.1	BuscaTarjeta()	32
4.6.2.2	CancelaReproduccion()	33
4.6.2.3	clean_list_files()	34
4.6.2.4	ComienzaReproduccion()	34

4.6.2.5	ComienzaSistema()	34
4.6.2.6	CompruebaComienzo()	35
4.6.2.7	CompruebaFinalReproduccion()	35
4.6.2.8	CompruebaTarjeta()	36
4.6.2.9	ConfiguracionCorrecta()	36
4.6.2.10	ConfiguraTarjeta()	37
4.6.2.11	DescartaTarjeta()	38
4.6.2.12	EsperaTargeta()	39
4.6.2.13	FinalizaReproduccion()	39
4.6.2.14	get_last_file()	40
4.6.2.15	get_list_files()	40
4.6.2.16	get_next_file()	41
4.6.2.17	get_number_files()	41
4.6.2.18	ISR()	42
4.6.2.19	killRFID()	43
4.6.2.20	launchRFID()	43
4.6.2.21	LeerTarjeta()	43
4.6.2.22	lp()	44
4.6.2.23	menu_display_stepper_plus()	44
4.6.2.24	new_list_files()	45
4.6.2.25	select_mode()	45
4.6.2.26	TarjetaDisponible()	46
4.6.2.27	TarjetaNoDisponible()	46
4.6.2.28	TarjetaNoValida()	47
4.6.2.29	TarjetaValida()	47
4.6.2.30	UUID_2_int()	48
4.6.3	Variable Documentation	48
4.6.3.1	maquina_creada	48
4.6.3.2	seqA	49
4.6.3.3	seqB	49

4.6.3.4	transition_table_rfid	49
4.6.3.5	UUID	49
4.7	fsm_rfid.h File Reference	50
4.7.1	Typedef Documentation	51
4.7.1.1	list_files_t	51
4.7.2	Enumeration Type Documentation	51
4.7.2.1	flags_rfid	51
4.7.2.2	rfid_states	52
4.7.3	Function Documentation	52
4.7.3.1	CancelaReproduccion()	52
4.7.3.2	ComienzaReproduccion()	53
4.7.3.3	ComienzaSistema()	53
4.7.3.4	CompruebaComienzo()	54
4.7.3.5	CompruebaFinalReproduccion()	54
4.7.3.6	CompruebaTarjeta()	55
4.7.3.7	DescartaTarjeta()	55
4.7.3.8	EsperaTargeta()	55
4.7.3.9	FinalizaReproduccion()	56
4.7.3.10	launchRFID()	56
4.7.3.11	LeerTarjeta()	56
4.7.3.12	TarjetaDisponible()	57
4.7.3.13	TarjetaNoDisponible()	57
4.7.3.14	TarjetaNoValida()	58
4.7.3.15	TarjetaValida()	58
4.7.4	Variable Documentation	59
4.7.4.1	fsm_rfid	59
4.7.4.2	thread	59
4.7.4.3	UUID	59
4.8	InterruptSM.c File Reference	60
4.8.1	Macro Definition Documentation	60

4.8.1.1	THRESHOLD_HIGH	60
4.8.1.2	THRESHOLD_LOW	60
4.8.2	Function Documentation	61
4.8.2.1	attachIsr()	61
4.8.2.2	deleteIsr()	61
4.8.2.3	loop()	62
4.9	InterruptSM.h File Reference	63
4.9.1	Typedef Documentation	63
4.9.1.1	call_back	63
4.9.1.2	ISR_Typ_	64
4.9.2	Enumeration Type Documentation	64
4.9.2.1	event	64
4.9.3	Function Documentation	64
4.9.3.1	attachIsr()	64
4.9.3.2	deleteIsr()	65
4.9.4	Variable Documentation	65
4.9.4.1	atachPin	65
4.9.4.2	threads	66
4.10	mutex.c File Reference	66
4.10.1	Function Documentation	66
4.10.1.1	lock()	66
4.10.1.2	unlock()	67
4.11	mutex.h File Reference	68
4.11.1	Function Documentation	68
4.11.1.1	lock()	68
4.11.1.2	unlock()	68
4.12	piMusicBox_2.c File Reference	69
4.12.1	Function Documentation	69
4.12.1.1	callback()	69
4.12.1.2	main()	70

4.12.2	Variable Documentation	70
4.12.2.1	k	70
4.13	piMusicBox_2.h File Reference	71
4.14	player.c File Reference	71
4.14.1	Function Documentation	72
4.14.1.1	carga_bff1()	72
4.14.1.2	carga_bff2()	72
4.14.1.3	Final_Melodia()	73
4.14.1.4	func()	73
4.14.1.5	Iniciliza_player()	74
4.14.1.6	launchPlayer()	75
4.14.1.7	new_buffer()	76
4.14.1.8	output()	76
4.14.2	Variable Documentation	77
4.14.2.1	err	77
4.14.2.2	fp	77
4.14.2.3	mad_frame	77
4.14.2.4	mad_stream	77
4.14.2.5	mad_synth	78
4.14.2.6	outputParameters	78
4.14.2.7	stream	78
4.14.2.8	transition_table_player	78
4.15	player.h File Reference	79
4.15.1	Macro Definition Documentation	79
4.15.1.1	FRAMES_PER_BUFFER	80
4.15.1.2	SAMPLE_RATE	80
4.15.2	Typedef Documentation	80
4.15.2.1	BUFFERS_T	80
4.15.2.2	fsm_audio_controller_t	80
4.15.3	Enumeration Type Documentation	80

4.15.3.1	fsm_states	80
4.15.4	Function Documentation	81
4.15.4.1	launchPlayer()	81
4.16	tipos.h File Reference	81
4.16.1	Macro Definition Documentation	82
4.16.1.1	FLAG_BFF1_END	82
4.16.1.2	FLAG_BFF2_END	83
4.16.1.3	FLAG_END	83
4.16.1.4	FLAG_IRQ_STEPPER_CONTINUE	83
4.16.1.5	FLAG_IRQ_STEPPER_DIR	83
4.16.1.6	FLAG_IRQ_STEPPER_SELECT	83
4.16.1.7	FLAG_NOTA_TIMEOUT	84
4.16.1.8	FLAG_PLAYER_END	84
4.16.1.9	FLAG_PLAYER_START	84
4.16.1.10	FLAG_PLAYER_STOP	84
4.16.1.11	FLAG_QUIT	84
4.16.1.12	FLAG_START	84
4.16.2	Variable Documentation	85
4.16.2.1	flag_fsm	85
4.16.2.2	flag_rfid	85
4.16.2.3	flags_player	85
4.16.2.4	num_file	85
4.16.2.5	song_name	85
4.16.2.6	stepper_irq_flag	86
4.17	tone.c File Reference	86
4.17.1	Function Documentation	86
4.17.1.1	toggle()	86
4.17.1.2	tone_init()	87
4.17.1.3	tone_stop()	87
4.17.1.4	tone_write()	88
4.17.2	Variable Documentation	88
4.17.2.1	PIN	88
4.18	tone.h File Reference	88
4.18.1	Function Documentation	88
4.18.1.1	tone_init()	89
4.18.1.2	tone_stop()	89
4.18.1.3	tone_write()	89

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

BUFFERS	
Estructura que almacena los buffers del Ping Pong	5
fsm_	7
fsm_audio_controller	
Estructura de la maquina de estados player	8
fsm_trans_	9
ISR_Typ	11
list_files	12
TipoSistema	13

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

dbcontroller.c	15
dbcontroller.h	21
defines.h	25
fsm.c	26
fsm.h	27
fsm_rfid.c	30
fsm_rfid.h	50
InterruptSM.c	60
InterruptSM.h	63
mutex.c	66
mutex.h	68
piMusicBox_2.c	69
piMusicBox_2.h	71
player.c	71
player.h	79
tipos.h	81
tone.c	86
tone.h	88

Chapter 3

Data Structure Documentation

3.1 BUFFERS Struct Reference

Estructura que almacena los buffers del Ping Pong.

```
#include <player.h>
```

Data Fields

- int * [buff1_r](#)
Buffer1 del canal derecho.
- int * [buff2_r](#)
Buffer2 del canal derecho.
- int * [buff1_l](#)
Buffer1 del canal izquierdo.
- int * [buff2_l](#)
Buffer1 del canal izquierdo.
- uint8_t [currentBuffer](#)
Buffer actual.
- int [lengthBuffer](#)
Longitud de los buffers.
- int [sampleReaded](#)
Puntero de muestra.
- volatile uint8_t [flags](#)
Flags asociados al Buffer.

3.1.1 Detailed Description

Estructura que almacena los buffers del Ping Pong.

Definition at line 42 of file player.h.

3.1.2 Field Documentation

3.1.2.1 buff1_l

```
int* buff1_l
```

Buffer1 del canal izquierdo.

Definition at line 47 of file player.h.

3.1.2.2 buff1_r

```
int* buff1_r
```

Buffer1 del canal derecho.

Definition at line 44 of file player.h.

3.1.2.3 buff2_l

```
int* buff2_l
```

Buffer1 del canal izquierdo.

Definition at line 48 of file player.h.

3.1.2.4 buff2_r

```
int* buff2_r
```

Buffer2 del canal derecho.

Definition at line 45 of file player.h.

3.1.2.5 currentBuffer

```
uint8_t currentBuffer
```

Buffer actual.

Definition at line 50 of file player.h.

3.1.2.6 flags

```
volatile uint8_t flags
```

Flags asociados al Buffer.

Definition at line 54 of file player.h.

3.1.2.7 lengthBuffer

```
int lengthBuffer
```

Longitud de los buffers.

Definition at line 51 of file player.h.

3.1.2.8 sampleReaded

```
int sampleReaded
```

Puntero de muestra.

Definition at line 52 of file player.h.

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

- [player.h](#)

3.2 fsm_ Struct Reference

```
#include <fsm.h>
```

Data Fields

- int [current_state](#)
- [fsm_trans_t](#) * [tt](#)
- void * [user_data](#)

3.2.1 Detailed Description

Definition at line 25 of file fsm.h.

3.2.2 Field Documentation

3.2.2.1 `current_state`

```
int current_state
```

Definition at line 26 of file fsm.h.

3.2.2.2 `tt`

```
fsm_trans_t* tt
```

Definition at line 27 of file fsm.h.

3.2.2.3 `user_data`

```
void* user_data
```

Definition at line 28 of file fsm.h.

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

- [fsm.h](#)

3.3 fsm_audio_controller Struct Reference

Estructura de la maquina de estados player.

```
#include <player.h>
```

Data Fields

- [fsm_t](#) * [fsm](#)
Puntero comun de maquina de estados.
- [TipoSistema](#) * [tipo_sistema](#)
Puntero a tipo sistema -> en desuso.
- [BUFFERS_T](#) * [buffer](#)
Puntero a los buffers.

3.3.1 Detailed Description

Estructura de la maquina de estados player.

Definition at line 61 of file player.h.

3.3.2 Field Documentation

3.3.2.1 buffer

`BUFFERS_T* buffer`

Puntero a los buffers.

Definition at line 65 of file player.h.

3.3.2.2 fsm

`fsm_t* fsm`

Puntero comun de maquina de estados.

Definition at line 63 of file player.h.

3.3.2.3 tipo_sistema

`TipoSistema* tipo_sistema`

Puntero a tipo sistema -> en desuso.

Definition at line 64 of file player.h.

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

- [player.h](#)

3.4 fsm_trans_ Struct Reference

```
#include <fsm.h>
```

Data Fields

- [int orig_state](#)
- [fsm_input_func_t in](#)
- [int dest_state](#)
- [fsm_output_func_t out](#)

3.4.1 Detailed Description

Definition at line 18 of file fsm.h.

3.4.2 Field Documentation

3.4.2.1 dest_state

```
int dest_state
```

Definition at line 21 of file fsm.h.

3.4.2.2 in

```
fsm_input_func_t in
```

Definition at line 20 of file fsm.h.

3.4.2.3 orig_state

```
int orig_state
```

Definition at line 19 of file fsm.h.

3.4.2.4 out

```
fsm_output_func_t out
```

Definition at line 22 of file fsm.h.

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

- [fsm.h](#)

3.5 ISR_Typ Struct Reference

```
#include <InterruptSM.h>
```

Data Fields

- `uint8_t` [pin](#)
- `uint8_t` [event](#)
- `call_back` [callback](#)

3.5.1 Detailed Description

Definition at line 24 of file InterruptSM.h.

3.5.2 Field Documentation

3.5.2.1 `callback`

`call_back` [callback](#)

Definition at line 27 of file InterruptSM.h.

3.5.2.2 `event`

`uint8_t` [event](#)

Definition at line 26 of file InterruptSM.h.

3.5.2.3 `pin`

`uint8_t` [pin](#)

Definition at line 25 of file InterruptSM.h.

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

- [InterruptSM.h](#)

3.6 list_files Struct Reference

```
#include <fsm_rfid.h>
```

Data Fields

- int [num_files](#)
Numero de ficheros totales.
- int [current_file](#)
Puntero fichero muestra.
- int [select_file](#)
Puntero fichero Seleccionado.
- char ** [name_file](#)
Lista de Strings.

3.6.1 Detailed Description

Definition at line 23 of file fsm_rfid.h.

3.6.2 Field Documentation

3.6.2.1 current_file

```
int current_file
```

Puntero fichero muestra.

Definition at line 25 of file fsm_rfid.h.

3.6.2.2 name_file

```
char** name_file
```

Lista de Strings.

Definition at line 27 of file fsm_rfid.h.

3.6.2.3 num_files

```
int num_files
```

Numero de ficheros totales.

Definition at line 24 of file fsm_rfid.h.

3.6.2.4 select_file

```
int select_file
```

Puntero fichero Seleccionado.

Definition at line 26 of file fsm_rfid.h.

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

- [fsm_rfid.h](#)

3.7 TipoSistema Struct Reference

```
#include <tipos.h>
```

3.7.1 Detailed Description

Definition at line 47 of file tipos.h.

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

- [tipos.h](#)

Chapter 4

File Documentation

4.1 dbcontroller.c File Reference

```
#include "dbcontroller.h"
#include <sqlite3.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

Macros

- #define [SQL_QUERY_CREATE_DB](#) "CREATE TABLE %s(USERID INT PRIMARY KEY NOT NULL,SONG TEXT NOT NULL);"
- #define [SQL_QUERY_INSERT_DB](#) "INSERT INTO %s (USERID,SONG) VALUES (%d,'%s'); "
- #define [SQL_QUERY_CHECK_DB](#) "SELECT name FROM sqlite_master WHERE type='table' AND name='%s';"
- #define [SQL_QUERY_SELECT_DB](#) "SELECT SONG FROM %s WHERE USERID = %d;"
- #define [SQL_QUERY_GET_NUMBER_OF_ELEMENTS](#) "SELECT count(*) FROM %s WHERE USERID = %d;"
- #define [SQL_TABLE_NAME](#) "TARJETAS"
- #define [SQL_QUERY_MAX_SIZE](#) 500

Functions

- void [db_create_tables](#) (sqlite3 *db)
Comprueba si es una base de datos nueva y en caso de serlo crea las tablas necesarias La funcion comprueba si existen la tabla especificada intentando hacer la query de creación, si dicha query se ha ejecutado correctamente, quiere decir que la base de datos es nueva, por lo que se crean las tablas necesarias para adecuarla al modelo de datos. Si la ejecución da error, quiere decir que la base de datos ya existía y no tienen que sobrescribirse las tablas.
- sqlite3 * [db_load](#) (char *file_dir)
- int [db_check](#) (sqlite3 *db)
Ejecuta una query para ver si existe la tabla con el valor especificado.
- int [db_insert](#) (sqlite3 *db, int user_id, char *[song_name](#))
Ejecuta la query para insertar en la base de datos.
- char * [db_get_song_name](#) (sqlite3 *db, int user_id)
Ejecuta la query de busqueda del nombre de la canción dado un ID Hace una query de busqueda del fichero que esta asociado a un ID.
- void [db_free_song_name](#) (char *name)
Libera memoria del reservado para el nombre de la cancion Binding de la función free.
- void [db_close](#) (sqlite3 *db)
Cierra la base de datos y libera la memoria asignada al puntero.

4.1.1 Macro Definition Documentation

4.1.1.1 SQL_QUERY_CHECK_DB

```
#define SQL_QUERY_CHECK_DB "SELECT name FROM sqlite_master WHERE type='table' AND name='%s';"
```

Definition at line 19 of file dbcontroller.c.

4.1.1.2 SQL_QUERY_CREATE_DB

```
#define SQL_QUERY_CREATE_DB "CREATE TABLE %s (USERID INT PRIMARY KEY NOT NULL, SONG TEXT NOT NULL);"
```

Definition at line 17 of file dbcontroller.c.

4.1.1.3 SQL_QUERY_GET_NUMBER_OF_ELEMENTS

```
#define SQL_QUERY_GET_NUMBER_OF_ELEMENTS "SELECT count(*) FROM %s WHERE USERID = %d;"
```

Definition at line 21 of file dbcontroller.c.

4.1.1.4 SQL_QUERY_INSERT_DB

```
#define SQL_QUERY_INSERT_DB "INSERT INTO %s (USERID, SONG) VALUES (%d, '%s');"
```

Definition at line 18 of file dbcontroller.c.

4.1.1.5 SQL_QUERY_MAX_SIZE

```
#define SQL_QUERY_MAX_SIZE 500
```

Definition at line 25 of file dbcontroller.c.

4.1.1.6 SQL_QUERY_SELECT_DB

```
#define SQL_QUERY_SELECT_DB "SELECT SONG FROM %s WHERE USERID = %d;"
```

Definition at line 20 of file dbcontroller.c.

4.1.1.7 SQL_TABLE_NAME

```
#define SQL_TABLE_NAME "TARJETAS"
```

Definition at line 23 of file dbcontroller.c.

4.1.2 Function Documentation

4.1.2.1 db_check()

```
int db_check (  
    sqlite3 * db )
```

Ejecuta una query para ver si existe la tabla con el valor especificado.

Parameters

<i>db</i>	Sqlite3 puntero de la estructura cargada
-----------	--

Returns

int controll code

Definition at line 63 of file dbcontroller.c.

```
63         {  
64     int rc;  
65     char* sql = (char*)malloc(SQL_QUERY_MAX_SIZE*sizeof(char));  
66     char* zErrMsg = 0;  
67     const char* data = "Callback function called";  
68     sprintf(sql, SQL_QUERY_CHECK_DB, SQL_TABLE_NAME);  
69  
70     rc = sqlite3_exec(db, sql, callback, (void*) data, &zErrMsg);  
71     if (rc != SQLITE_OK) {  
72         fprintf(stderr, "Error query busqueda tablas: %s", zErrMsg);  
73         sqlite3_free(zErrMsg);  
74         return SQL_OPERATION_ERR;  
75     } else {  
76         fprintf(stdout, "Todo correcto buscando tablas\n");  
77     }  
78     free(sql);  
79     return SQL_OPERATION_OK;  
80 }
```

4.1.2.2 db_close()

```
void db_close (
    sqlite3 * db )
```

Cierra la base de datos y libera la memoria asignada al puntero.

Parameters

<i>db</i>	
-----------	--

Definition at line 203 of file dbcontroller.c.

```
203         {
204     sqlite3_close(db);
205 }
```

4.1.2.3 db_create_tables()

```
void db_create_tables (
    sqlite3 * db )
```

Comprueba si es una base de datos nueva y en caso de serlo crea las tablas necesarias La funcion comprueba si existen la tabla especificada intentando hacer la query de creación, si dicha query se ha ejecutado correctamente, quiere decir que la base de datos es nueva, por lo que se crean las tablas necesarias para adecuarla al modelo de datos. Si la ejecución da error, quiere decir que la base de datos ya existía y no tienen que sobrescribirse las tablas.

Parameters

<i>db</i>	Sqlite3 puntero de la estructura cargada
-----------	--

Definition at line 94 of file dbcontroller.c.

```
94         {
95     int rc;
96     char* sql = (char*)malloc(SQL_QUERY_MAX_SIZE*sizeof(char));
97     char* zErrMsg = 0;
98     //Formateamos la query con sprintf para permitir genericos
99     sprintf(sql,SQL_QUERY_CREATE_DB,SQL_TABLE_NAME);
100    /*
101     * Si la query me da error, es porque esa tabla ya existe,
102     * por lo que la creacion se ignora y no afecta a los datos
103     */
104    rc = sqlite3_exec(db, sql, NULL, 0, &zErrMsg);
105    if (rc != SQLITE_OK) {
106        //Si tengo error, la base ya tiene la tabla
107        fprintf(stderr, "Base de datos con datos, no creo tablas\n Por si acaso printeo el error %s \n",
zErrMsg);
108        //Libero memoria del error
109        sqlite3_free(zErrMsg);
110    } else {
111        /* Si no tengo error al inicializar la base de datos
112         * he creado una nueva, por lo que debo formatear las tablas
113         * acorde a lo esperado por la tarjeta
114         */
115        fprintf(stdout, "Base de datos nueva, creo las tablas\n");
```

```
116     }
117     //Libero memoria de la query
118     free(sql);
119 }
```

4.1.2.4 db_free_song_name()

```
void db_free_song_name (
    char * name )
```

Libera memoria del reservado para el nombre de la cancion Binding de la función free.

Parameters

<i>name</i>	string del que se quiere liberar memoria
-------------	--

Definition at line 195 of file dbcontroller.c.

```
195                                     {
196     free(name);
197 }
```

4.1.2.5 db_get_song_name()

```
char* db_get_song_name (
    sqlite3 * db,
    int user_id )
```

Ejecuta la query de busqueda del nombre de la canción dado un ID Hace una query de busqueda del fichero que esta asociado a un ID.

Parameters

<i>db</i>	Sqlite3 puntero de la estructura cargada
<i>user_id</i>	Id de la tarjeta

Returns

char* nombre de la canción, NULL

Definition at line 157 of file dbcontroller.c.

```
157                                     {
158     char *zErrMsg = 0;
159     int rc;
160     char* sql = (char*) malloc(SQL_QUERY_MAX_SIZE * sizeof(char));
```

```

161     char* name = (char*)malloc(sizeof(char)*200);
162     sprintf(sql,SQL_QUERY_GET_NUMBER_OF_ELEMENTS,
SQL_TABLE_NAME,user_id);
163     printf("%s",sql);
164     rc = sqlite3_exec(db, sql, db_callback_get_num_of_elm, name, &zErrMsg);
165     if (rc != SQLITE_OK) {
166         fprintf(stderr, "SQL error: %s\n", zErrMsg);
167         sqlite3_free(zErrMsg);
168         free(sql);
169         return NULL;
170     }
171     fprintf(stdout, "Lectura correcta de la base de datos\n");
172     if(*name == 0x30){
173         db_free_song_name(name);
174         return NULL;
175     }
176     sprintf(sql,SQL_QUERY_SELECT_DB,SQL_TABLE_NAME,user_id);
177     rc = sqlite3_exec(db, sql, db_callback_get_song_name, name, &zErrMsg);
178     if (rc != SQLITE_OK) {
179         fprintf(stderr, "SQL error: %s\n", zErrMsg);
180         sqlite3_free(zErrMsg);
181         free(sql);
182         return NULL;
183     }
184     fprintf(stdout, "Lectura correcta de la base de datos\n");
185     free(sql);
186     return name;
187 }

```

4.1.2.6 db_insert()

```

int db_insert (
    sqlite3 * db,
    int user_id,
    char * song_name )

```

Ejecuta la query para insertar en la base de datos.

Parameters

<i>db</i>	Sqlite3 puntero de la estructura cargada
<i>user_id</i>	Id que se quiere almacenar
<i>song_name</i>	Nombre del fichero asociado al ID

Returns

int Código de control

Definition at line 129 of file dbcontroller.c.

```

129                                     {
130     char *zErrMsg = 0;
131     int rc;
132     char* sql = (char*) malloc(SQL_QUERY_MAX_SIZE * sizeof(char));
133
134     sprintf(sql,SQL_QUERY_INSERT_DB,SQL_TABLE_NAME,user_id,
song_name);
135
136     rc = sqlite3_exec(db, sql, NULL, 0, &zErrMsg);
137     if (rc != SQLITE_OK) {
138         fprintf(stderr, "SQL error: %s\n", zErrMsg);
139         sqlite3_free(zErrMsg);
140         free(sql);
141         return SQLITE_OPERATION_ERR;

```



```

142     } else {
143         fprintf(stdout, "Escritura correcta en la base de datos\n");
144     }
145     free(sql);
146     return SQL_OPERATION_OK;
147 }

```

4.1.2.7 db_load()

```

sqlite3* db_load (
    char * file_dir )

```

Definition at line 42 of file dbcontroller.c.

```

42     {
43         sqlite3* db;
44         int rc;
45         if((rc = sqlite3_open(file_dir,&db)){
46             fprintf(stderr, "No se puede abrir la base de datos %s \n", sqlite3_errmsg(db));
47             return NULL;
48         }
49         fprintf(stdout, "Base de datos abierta \n");
50         /*Funcion de persistencia para que siempre que haya una base de datos
51          * se compruebe que sea correcta
52          */
53         db_create_tables(db);
54         return db;
55 }

```

4.2 dbcontroller.h File Reference

```
#include <sqlite3.h>
```

Enumerations

- enum [SQL_RESOLUTION](#) { [SQL_OPERATION_OK](#), [SQL_OPERATION_ERR](#) }
Enum de resultados posibles de ejecución.

Functions

- sqlite3 * [db_load](#) (char *file_dir)
- int [db_check](#) (sqlite3 *db)
Ejecuta una query para ver si existe la tabla con el valor especificado.
- int [db_insert](#) (sqlite3 *db, int user_id, char *song_name)
Ejecuta la query para insertar en la base de datos.
- char * [db_get_song_name](#) (sqlite3 *db, int user_id)
Ejecuta la query de busqueda del nombre de la canción dado un ID Hace una query de busqueda del fichero que esta asociado a un ID.
- void [db_close](#) (sqlite3 *db)
Cierra la base de datos y libera la memoria asignada al puntero.

4.2.1 Enumeration Type Documentation

4.2.1.1 SQL_RESOLUTION

enum [SQL_RESOLUTION](#)

Enum de resultados posibles de ejecución.

Enumerator

SQL_OPERATION_OK	Operacion correcta.
SQL_OPERATION_ERR	Operacion erronea.

Definition at line 18 of file dbcontroller.h.

```

18         {
19     SQL\_OPERATION\_OK,
20     SQL\_OPERATION\_ERR
21 };

```

4.2.2 Function Documentation

4.2.2.1 db_check()

```

int db_check (
    sqlite3 * db )

```

Ejecuta una query para ver si existe la tabla con el valor especificado.

Parameters

<i>db</i>	Sqlite3 puntero de la estructura cargada
-----------	--

Returns

int controll code

Definition at line 63 of file dbcontroller.c.

```

63         {
64     int rc;
65     char* sql = (char*)malloc(SQL_QUERY_MAX_SIZE*sizeof(char));
66     char* zErrMsg = 0;
67     const char* data = "Callback function called";
68     sprintf(sql, SQL\_QUERY\_CHECK\_DB, SQL\_TABLE\_NAME);
69

```

```

70     rc = sqlite3_exec(db, sql, callback, (void*) data, &zErrMsg);
71     if (rc != SQLITE_OK) {
72         fprintf(stderr, "Error query busqueda tablas: %s", zErrMsg);
73         sqlite3_free(zErrMsg);
74         return SQLITE_OPERATION_ERR;
75     } else {
76         fprintf(stdout, "Todo correcto buscando tablas\n");
77     }
78     free(sql);
79     return SQLITE_OPERATION_OK;
80 }

```

4.2.2.2 db_close()

```

void db_close (
    sqlite3 * db )

```

Cierra la base de datos y libera la memoria asignada al puntero.

Parameters

<i>db</i>	
-----------	--

Definition at line 203 of file dbcontroller.c.

```

203     {
204         sqlite3_close(db);
205     }

```

4.2.2.3 db_get_song_name()

```

char* db_get_song_name (
    sqlite3 * db,
    int user_id )

```

Ejecuta la query de busqueda del nombre de la canción dado un ID Hace una query de busqueda del fichero que esta asociado a un ID.

Parameters

<i>db</i>	Sqlite3 puntero de la estructura cargada
<i>user_id</i>	Id de la tarjeta

Returns

char* nombre de la canción, NULL

Definition at line 157 of file dbcontroller.c.

```

157                                     {
158     char *zErrMsg = 0;
159     int rc;
160     char* sql = (char*) malloc(SQL_QUERY_MAX_SIZE * sizeof(char));
161     char* name = (char*)malloc(sizeof(char)*200);
162     sprintf(sql,SQL_QUERY_GET_NUMBER_OF_ELEMENTS,
SQL_TABLE_NAME,user_id);
163     printf("%s",sql);
164     rc = sqlite3_exec(db, sql, db_callback_get_num_of_elm, name, &zErrMsg);
165     if (rc != SQLITE_OK) {
166         fprintf(stderr, "SQL error: %s\n", zErrMsg);
167         sqlite3_free(zErrMsg);
168         free(sql);
169         return NULL;
170     }
171     fprintf(stdout, "Lectura correcta de la base de datos\n");
172     if(*name) == 0x30){
173         db_free_song_name(name);
174         return NULL;
175     }
176     sprintf(sql,SQL_QUERY_SELECT_DB,SQL_TABLE_NAME,user_id);
177     rc = sqlite3_exec(db, sql, db_callback_get_song_name, name, &zErrMsg);
178     if (rc != SQLITE_OK) {
179         fprintf(stderr, "SQL error: %s\n", zErrMsg);
180         sqlite3_free(zErrMsg);
181         free(sql);
182         return NULL;
183     }
184     fprintf(stdout, "Lectura correcta de la base de datos\n");
185     free(sql);
186     return name;
187 }

```

4.2.2.4 db_insert()

```

int db_insert (
    sqlite3 * db,
    int user_id,
    char * song_name )

```

Ejecuta la query para insertar en la base de datos.

Parameters

<i>db</i>	Sqlite3 puntero de la estructura cargada
<i>user_id</i>	Id que se quiere almacenar
<i>song_name</i>	Nombre del fichero asociado al ID

Returns

int Codigo de control

Definition at line 129 of file dbcontroller.c.

```

129                                     {
130     char *zErrMsg = 0;
131     int rc;
132     char* sql = (char*) malloc(SQL_QUERY_MAX_SIZE * sizeof(char));
133
134     sprintf(sql,SQL_QUERY_INSERT_DB,SQL_TABLE_NAME,user_id,
song_name);
135
136     rc = sqlite3_exec(db, sql, NULL, 0, &zErrMsg);
137     if (rc != SQLITE_OK) {

```

```

138         fprintf(stderr, "SQL error: %s\n", zErrMsg);
139         sqlite3_free(zErrMsg);
140         free(sql);
141         return SQL_OPERATION_ERR;
142     } else {
143         fprintf(stdout, "Escritura correcta en la base de datos\n");
144     }
145     free(sql);
146     return SQL_OPERATION_OK;
147 }

```

4.2.2.5 db_load()

```

sqlite3* db_load (
    char * file_dir )

```

Definition at line 42 of file dbcontroller.c.

```

42         {
43     sqlite3* db;
44     int rc;
45     if((rc = sqlite3_open(file_dir,&db)){
46         fprintf(stderr,"No se puede abrir la base de datos %s \n",sqlite3_errmsg(db));
47         return NULL;
48     }
49     fprintf(stdout,"Base de datos abierta \n");
50     /*Funcion de persistencia para que siempre que haya una base de datos
51      * se compruebe que sea correcta
52      */
53     db_create_tables(db);
54     return db;
55 }

```

4.3 defines.h File Reference

Macros

- #define [bcm_spi](#)
- #define [CANCION_ACABADA](#) 1
- #define [CANCION_NOACABADA](#) 0;
- #define [use_bcm](#)
- #define [PIN_PWM](#) 18

4.3.1 Macro Definition Documentation

4.3.1.1 bcm_spi

```
#define bcm_spi
```

Definition at line 18 of file defines.h.

4.3.1.2 CACION_ACABADA

```
#define CACION_ACABADA 1
```

Definition at line 21 of file defines.h.

4.3.1.3 CACION_NOACABADA

```
#define CACION_NOACABADA 0;
```

Definition at line 22 of file defines.h.

4.3.1.4 PIN_PWM

```
#define PIN_PWM 18
```

Definition at line 29 of file defines.h.

4.3.1.5 use_bcm

```
#define use_bcm
```

Definition at line 24 of file defines.h.

4.4 fsm.c File Reference

```
#include "fsm.h"
```

Functions

- [fsm_t * fsm_new](#) ([fsm_trans_t](#) *tt, void *user_data)
- int [fsm_delete](#) ([fsm_t](#) *fsm)
- void [fsm_fire](#) ([fsm_t](#) *this)

4.4.1 Function Documentation

4.4.1.1 fsm_delete()

```
int fsm_delete (
    fsm_t * fsm )
```

Definition at line 18 of file fsm.c.

```
18     {
19     free(fsm);
20     return 1;
21 }
```

4.4.1.2 fsm_fire()

```
void fsm_fire (
    fsm_t * this )
```

Definition at line 23 of file fsm.c.

```
23     {
24     fsm_trans_t* t;
25     for (t = this->tt; t->orig_state >= 0; ++t) {
26         if ((this->current_state == t->orig_state) && t->in(this)) {
27             this->current_state = t->dest_state;
28             if (t->out)
29                 t->out(this);
30             break;
31         }
32     }
33 }
```

4.4.1.3 fsm_new()

```
fsm_t* fsm_new (
    fsm_trans_t * tt,
    void * user_data )
```

Definition at line 10 of file fsm.c.

```
10     {
11     fsm_t* new_fsm = (fsm_t*)malloc(sizeof (fsm_t));
12     new_fsm->tt = tt;
13     new_fsm->current_state = 0;
14     new_fsm->user_data = user_data;
15     return new_fsm;
16 }
```

4.5 fsm.h File Reference

```
#include <stdlib.h>
```

Data Structures

- struct [fsm_trans_](#)
- struct [fsm_](#)

Typedefs

- typedef struct [fsm_ fsm_t](#)
- typedef int(* [fsm_input_func_t](#)) ([fsm_t](#) *)
- typedef void(* [fsm_output_func_t](#)) ([fsm_t](#) *)
- typedef struct [fsm_trans_ fsm_trans_t](#)

Functions

- [fsm_t](#) * [fsm_new](#) ([fsm_trans_t](#) *tt, void *user_data)
- void [fsm_fire](#) ([fsm_t](#) *fsm)
- int [fsm_delete](#) ([fsm_t](#) *fsm)

4.5.1 Typedef Documentation

4.5.1.1 fsm_input_func_t

```
typedef int(* fsm_input_func_t) (fsm\_t *)
```

Definition at line 15 of file fsm.h.

4.5.1.2 fsm_output_func_t

```
typedef void(* fsm_output_func_t) (fsm\_t *)
```

Definition at line 16 of file fsm.h.

4.5.1.3 fsm_t

```
typedef struct fsm\_ fsm\_t
```

Definition at line 13 of file fsm.h.

4.5.1.4 fsm_trans_t

```
typedef struct fsm_trans_ fsm_trans_t
```

4.5.2 Function Documentation

4.5.2.1 fsm_delete()

```
int fsm_delete (
    fsm_t * fsm )
```

Definition at line 18 of file fsm.c.

```
18                                     {
19     free(fsm);
20     return 1;
21 }
```

4.5.2.2 fsm_fire()

```
void fsm_fire (
    fsm_t * fsm )
```

Definition at line 23 of file fsm.c.

```
23                                     {
24     fsm_trans_t* t;
25     for (t = this->tt; t->orig_state >= 0; ++t) {
26         if ((this->current_state == t->orig_state) && t->in(this)) {
27             this->current_state = t->dest_state;
28             if (t->out)
29                 t->out(this);
30             break;
31         }
32     }
33 }
```

4.5.2.3 fsm_new()

```
fsm_t* fsm_new (
    fsm_trans_t * tt,
    void * user_data )
```

Definition at line 10 of file fsm.c.

```
10                                     {
11     fsm_t* new_fsm = (fsm_t*)malloc(sizeof (fsm_t));
12     new_fsm->tt = tt;
13     new_fsm->current_state = 0;
14     new_fsm->user_data = user_data;
15     return new_fsm;
16 }
```

4.6 fsm_rfid.c File Reference

```
#include <stdio.h>
#include <dirent.h>
#include <sqlite3.h>
#include <stdlib.h>
#include <string.h>
#include "dbcontroller.h"
#include "fsm_rfid.h"
#include "mutex.h"
#include "menu_lcd.h"
#include "InterruptSM.h"
```

Macros

- `#define DB_NAME "canciones_db.db"`
- `#define PIN_A 17`
- `#define PIN_B 27`
- `#define PIN_C 22`

Functions

- void `ConfiguraTarjeta (fsm_t *fsm)`
Configura la tarjeta en la base de datos Muestra un menu por pantalla e inicializa las interrupciones del, encoder para que funcionen solamente en este punto del programa.
- void `ConfiguracionCorrecta ()`
Desinicializacion tras la configuracion de una tarjeta Elimina todas las interrupciones asociadas al encoder e imprime por el lcd el mensaje de final de la configuracion.
- void `BuscaTarjeta (fsm_t *fsm)`
Funcion de Búsqueda de la tarjeta en la Db Busca si existe la tarjeta en la base de datos, en funcion de este resultado activa los flags de FLAG_CARD_EXIST o no.
- int `UUID_2_int ()`
Conversor de Int Array a int del UUID.
- void `menu_display_stepper_plus (list_files_t *lista)`
Refresco del LCD con los parametros de la lista Funcion de manejo del menu que se actualiza en funcion de lo que se haya operado en la funcion lista, para poder mostrar un menu rotativo bidireccional por el LCD. Anade en la primera linea del LCD el prompt de seleccion.
- char * `get_next_file (list_files_t *lista)`
Se mueve una posicion hacia delante el puntero de fichero actual.
- `list_files_t * get_list_files (char *route)`
Crea una estructura de tipo lista en el directorio pasado como parametro Reserva memoria para una estructura de tipo lista de ficheros y la rellena con los datos del directorio que se esta escaneando.
- `list_files_t * new_list_files (int num_files)`
Devuelve una estructura con memoria reservada y parametros por defecto.
- int `get_number_files (char *route)`
Cuenta el numero de ficheros que hay en la ruta.
- void `ISR (int event)`
Registra las interrupciones del Encoder.
- void `select_mode (int event)`
Callback de boton del encoder Activa el flag de seleccion del encoder.
- int `CompruebaComienzo (fsm_t *fsm)`

- Funcion de arranque del sistema devuelve siempre 1.*

 - int [CompruebaFinalReproduccion](#) (fsm_t *fsm)
Condición de paso si Ha acabado la reproducción de la cancion.
 - int [TarjetaDisponible](#) (fsm_t *fsm)
Condición de paso si hay una tarjeta insertada en el optoacoplador.
 - int [TarjetaNoDisponible](#) (fsm_t *fsm)
Condición de paso si no hay una tarjeta insertada en el optoacoplador.
 - int [TarjetaValida](#) (fsm_t *fsm)
Condición de paso si la tarjeta es RFID.
 - int [TarjetaNoValida](#) (fsm_t *fsm)
Condición de paso si la tarjeta no es RFID.
 - void [EsperaTarjeta](#) (fsm_t *fsm)
Funcion de espera de la tarjeta mientras no haya una en el optoacoplador.
 - void [DescartaTarjeta](#) (fsm_t *fsm)
Altera el flag de tarjeta insertada.
 - void [LeerTarjeta](#) (fsm_t *fsm)
Lee la tarjeta, activa el flag de tarjeta valida Lee la tarjeta, almacena en una variable privada del fichero el UUID de la misma. Activa el flag de tarjeta valida.
 - void [ComienzaReproduccion](#) (fsm_t *fsm)
Activa el flag de inicio de reproduccion de la maquina de estados de player.
 - void [CancelaReproduccion](#) (fsm_t *fsm)
Activa el flag de inicio de stop de la maquina de estados de player.
 - void [CompruebaTarjeta](#) (fsm_t *fsm)
Funcion de espera de comprueba tarjeta.
 - void [FinalizaReproduccion](#) (fsm_t *fsm)
Funcion de finalizacion de reproduccion.
 - void [ComienzaSistema](#) (fsm_t *fsm)
Funcion de Inicizalizacion del sistema.
 - void [lp](#) (void *userData)
Funcion de loop infinito de la maquina de estados.
 - void [launchRFID](#) ()
Funcion de creacion de la maquina de estados Si la maquina de estados ya esta creada no se crea otra.
 - void [killRFID](#) ()
Mata la maquina de estados del rfid.
 - char * [get_last_file](#) (list_files_t *lista)
Se mueve una posicion hacia atrás el punero de fichero actual.
 - void [clean_list_files](#) (list_files_t *lista)
Destructor de una lista creada.

Variables

- uint8_t [UUID](#) [16]
Variable privada con el UUID de la ultima tarjeta.
- int [maquina_creada](#) = 0
Indica si la maquina de estados ha sido creada ya o no.
- fsm_trans_t [transition_table_rfid](#) []
- int [seqA](#)
- int [seqB](#)

4.6.1 Macro Definition Documentation

4.6.1.1 DB_NAME

```
#define DB_NAME "canciones_db.db"
```

Definition at line 20 of file fsm_rfid.c.

4.6.1.2 PIN_A

```
#define PIN_A 17
```

Definition at line 21 of file fsm_rfid.c.

4.6.1.3 PIN_B

```
#define PIN_B 27
```

Definition at line 22 of file fsm_rfid.c.

4.6.1.4 PIN_C

```
#define PIN_C 22
```

Definition at line 23 of file fsm_rfid.c.

4.6.2 Function Documentation

4.6.2.1 BuscaTarjeta()

```
void BuscaTarjeta (  
    fsm_t * fsm )
```

Funcion de Busqueda de la tarjeta en la Db Busca si existe la tarjeta en la base de datos, en funcion de este resultado activa los flags de FLAG_CARD_EXIST o no.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Definition at line 305 of file fsm_rfid.c.

```

306 {
307     //Iniciamos la base de datos
308     sqlite3 *db = db_load(DB_NAME);
309     lock(5);
310     printf("UUID: %d", UUID_2_int());
311     //Busca el nombre en la base de datos
312     song_name = db_get_song_name(db, UUID_2_int());
313     printf("SALGO \n");
314     delay(10);
315     printf("SongName: %s", song_name);
316     fflush(stdout);
317     //Si el puntero song_name es nulo no existe en la DB
318     if (song_name == NULL)
319     {
320         unlock(5);
321         printf("Error con los datos de la db \n");
322         //Activamos el flag de no existir
323         lock(0);
324         flag_rfid |= ~FLAG_CARD_EXIST;
325         unlock(0);
326         return;
327     }
328     unlock(5);
329     //Activamos el flag de existir
330     lock(0);
331     flag_rfid |= FLAG_CARD_EXIST;
332     unlock(0);
333     //Display por el LCD
334     db_close(db);
335 }
```

4.6.2.2 CancelaReproduccion()

```

void CancelaReproduccion (
    fsm_t * fsm )
```

Activa el flag de inicio de stop de la maquina de estados de player.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Definition at line 225 of file fsm_rfid.c.

```

226 {
227     lock(1);
228     flags_player |= FLAG_END;
229     unlock(1);
230 }
```

4.6.2.3 clean_list_files()

```
void clean_list_files (
    list_files_t * lista )
```

Destructor de una lista creada.

Parameters

<i>lista</i>	estructura que se quiere eliminar de la memoria
--------------	---

Definition at line 555 of file fsm_rfid.c.

```
556 {
557     int i = 0;
558     for (i = 0; i < lista->num_files; i++)
559     {
560         free(lista->name_file[i]);
561     }
562     free(lista->name_file);
563     free(lista);
564 }
```

4.6.2.4 ComienzaReproduccion()

```
void ComienzaReproduccion (
    fsm_t * fsm )
```

Activa el flag de inicio de reproduccion de la maquina de estados de player.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Definition at line 212 of file fsm_rfid.c.

```
213 {
214     menu_lcd_display("", "Playing: ", song_name, "");
215     lock(1);
216     flags_player = FLAG_START;
217     unlock(1);
218     printf("comienzas flag: %d", flags_player);
219 }
```

4.6.2.5 ComienzaSistema()

```
void ComienzaSistema (
    fsm_t * fsm )
```

Funcion de Inicizalizacion del sistema.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Definition at line 256 of file fsm_rfid.c.

```

257 {
258     printf("Arranca el RFID");
259     menu_lcd_display_clear(),
260     menu_lcd_display("PiMusicBox Player!", "", "", "");
261     fflush(stdout);
262     //Si no es la primera inicialización no ocurre nada aqui
263     if (RC522_Init() == STATUS_ERROR)
264         printf("ERROR! \n");
265     //Reset de todos los flags
266     lock(0);
267     flag_rfid = 0;
268     unlock(0);
269 }
```

4.6.2.6 CompruebaComienzo()

```

int CompruebaComienzo (
    fsm_t * fsm )
```

Funcion de arranque del sistema devuelve siempre 1.

Parameters

<i>fsm</i>	
------------	--

Returns

int, siempre 1

Definition at line 103 of file fsm_rfid.c.

```

104 {
105     return 1;
106 }
```

4.6.2.7 CompruebaFinalReproduccion()

```

int CompruebaFinalReproduccion (
    fsm_t * fsm )
```

Condición de paso si Ha acabado la reproducción de la cancion.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Returns

int Condición cumplida o no

Definition at line 113 of file fsm_rfid.c.

```
114 {  
115     lock(0);  
116     uint8_t tmp = (flag_rfid & FLAG_SYSTEM_END);  
117     unlock(0);  
118     return tmp;  
119 }
```

4.6.2.8 CompruebaTarjeta()

```
void CompruebaTarjeta (  
    fsm_t * fsm )
```

Funcion de espera de comprueba tarjeta.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Definition at line 236 of file fsm_rfid.c.

```
237 {  
238     return;  
239 }
```

4.6.2.9 ConfiguracionCorrecta()

```
void ConfiguracionCorrecta ( )
```

Desinicializacion tras la configuracion de una tarjeta Elimina todas las interrupciones asociadas al encoder e imprime por el lcd el mensaje de final de la configuracion.

Definition at line 399 of file fsm_rfid.c.

```
400 {  
401     printf("Configuracion Correcta \n");  
402     deleteIsr(PIN_A);  
403     deleteIsr(PIN_B);  
404     deleteIsr(PIN_C);  
405     menu_lcd_display_clear();  
406     menu_lcd_display("Finalizado!", " Retire la", "tarjeta", ":D");  
407 }
```


4.6.2.10 ConfiguraTarjeta()

```
void ConfiguraTarjeta (
    fsm_t * fsm )
```

Configura la tarjeta en la base de datos Muestra un menu por pantalla e inicializa las interrupciones del, encoder para que funcionen solamente en este punto del programa.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Definition at line 342 of file fsm_rfid.c.

```
343 {
344
345     printf("DATA -> PLAY \n");
346     printf("Configuracion de Tarjeta \n");
347     //Inicializa las interrupciones del Encoder
348     attachIsr(PIN_A, CHANGE, NULL, ISR);
349     attachIsr(PIN_B, CHANGE, NULL, ISR);
350     attachIsr(PIN_C, FALLIN_EDGE, NULL, select_mode);
351     //Imprime la captura inicial
352     menu_lcd_display("NOT CONFIGURED!", "turn to ", "configure", ":D");
353
354     stepper_irq_flag = 0;
355     //Cargo lista de archivos del directorio con la musica
356     list_files_t *lista = get_list_files("./musica");
357     //Bloqueo ejecucion hasta que se mueva el encoder una vez
358     while (!(stepper_irq_flag & FLAG_IRQ_STEPPER_CONTINUE))
359     ;
360     //Limpio los flags
361     lock(7);
362     stepper_irq_flag = 0;
363     unlock(7);
364     //Muestra la lista por pantalla
365     menu_display_stepper_plus(lista);
366     //Bloqueo ejecucion hasta que no hay una pulsacion
367     while (!(stepper_irq_flag & FLAG_IRQ_STEPPER_SELECT))
368     {
369         lock(7);
370         //Si detecto movimiento del encoder
371         if (stepper_irq_flag & FLAG_IRQ_STEPPER_CONTINUE)
372         {
373             lista->select_file++;
374             if (lista->select_file > lista->num_files)
375             {
376                 lista->select_file = 1;
377             }
378             //Refresco el display
379             menu_display_stepper_plus(lista);
380             stepper_irq_flag &= ~FLAG_IRQ_STEPPER_CONTINUE;
381
382             printf("Fichero %d", lista->select_file);
383             fflush(stdout);
384         }
385         unlock(7);
386     }
387     //Cargo base de datos y printeo pantalla final
388     sqlite3 *db = db_load(DB_NAME);
389     //Inserto en la db
390     db_insert(db, UUID_2_int(), lista->name_file[lista->
current_file - 1]);
391     ConfiguracionCorrecta();
392     db_close(db);
393 }
```

4.6.2.11 DescartaTarjeta()

```
void DescartaTarjeta (  
    fsm_t * fsm )
```

Altera el flag de tarjeta insertada.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Definition at line 186 of file fsm_rfid.c.

```
187 {  
188     lock(0);  
189     flag_rfid &= ~FLAG_CARD_IN;  
190     unlock(0);  
191 }
```

4.6.2.12 EsperaTargeta()

```
void EsperaTargeta (  
    fsm_t * fsm )
```

Funcion de espera de la tarjeta mientras no haya una en el optoacoplador.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Definition at line 177 of file fsm_rfid.c.

```
178 {  
179     return;  
180 }
```

4.6.2.13 FinalizaReproduccion()

```
void FinalizaReproduccion (  
    fsm_t * fsm )
```

Funcion de finalizacion de reproduccion.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Definition at line 245 of file fsm_rfid.c.

```
246 {  
247     lock(1);  
248     flags_player |= FLAG_END;  
249     unlock(1);  
250 }
```

4.6.2.14 get_last_file()

```
char* get_last_file (
    list_files_t * lista )
```

Se mueve una posicion hacia atrás el punero de fichero actual.

Parameters

<i>lista</i>	
--------------	--

Returns

char*

Definition at line 502 of file fsm_rfid.c.

```
503 {
504     return lista->current_file >= 0 ? lista->name_file[lista->
    current_file--] : NULL;
505 }
```

4.6.2.15 get_list_files()

```
list_files_t * get_list_files (
    char * route )
```

Crea una estructura de tipo lista en el directorio pasado como parametro Reserva memoria para una estructura de tipo lista de ficheros y la rellena con los datos del directorio que se esta escaneando.

Parameters

<i>route</i>	
--------------	--

Returns

list_files_t*

Definition at line 447 of file fsm_rfid.c.

```
448 {
449     DIR *dir;
450     struct dirent *ent;
451     list_files_t *lista;
452     //Inicializo la estructuar de datos
453     int num_files = get_number_files(route);
454     printf("nfiles = %d \n", num_files);
455     lista = new_list_files(num_files);
```

```

456     int i = 0;
457     //Abro el directorio de la ruta
458     if ((dir = opendir(route)) != NULL)
459     {
460         //Se ejecuta mientras haya algun fichero no leido
461         while ((ent = readdir(dir)) != NULL)
462         {
463             //Si es un tipo regular, fichero valido, lo añado a la lista
464             if (ent->d_type == DT_REG)
465             {
466                 printf("%s\n", ent->d_name);
467                 strcpy(lista->name_file[i++], ent->d_name);
468             }
469         }
470         //Cierro el directorio
471         closedir(dir);
472     }
473     else
474     {
475         printf("Error no se puede abrir el directorio");
476         return NULL;
477     }
478     printf("all ok \n");
479     return lista;
480 }

```

4.6.2.16 get_next_file()

```

char * get_next_file (
    list_files_t * lista )

```

Se mueve una posicion hacia delante el puntero de fichero actual.

Parameters

<i>lista</i>	
--------------	--

Returns

char*

Definition at line 487 of file fsm_rfid.c.

```

488 {
489
490     if (lista->current_file >= lista->num_files || lista->
current_file < 0)
491         lista->current_file = 0;
492     printf("current pos: %d \n", lista->current_file);
493     return lista->name_file[lista->current_file++];
494 }

```

4.6.2.17 get_number_files()

```

int get_number_files (
    char * route )

```

Cuenta el numero de ficheros que hay en la ruta.

Parameters

<i>route</i>	
--------------	--

Returns

int

Definition at line 513 of file fsm_rfid.c.

```

514 {
515     int file_count = 0;
516     DIR *dirp;
517     struct dirent *entry;
518
519     dirp = opendir(route); /* There should be error handling after this */
520     while ((entry = readdir(dirp)) != NULL)
521     {
522         if (entry->d_type == DT_REG)
523         { /* If the entry is a regular file */
524             file_count++;
525         }
526     }
527     closedir(dirp);
528     return file_count;
529 }

```

4.6.2.18 ISR()

```

void ISR (
    int event )

```

Registra las interrupciones del Encoder.

Parameters

<i>event</i>	evento que se ha producido
--------------	----------------------------

Definition at line 594 of file fsm_rfid.c.

```

595 {
596     //Muestreamos ambos canales
597     int A_val = bcm2835_gpio_lev(PIN_A);
598     int B_val = bcm2835_gpio_lev(PIN_B);
599     //Almaceno los valores muestreados
600     seqA <= 1;
601     seqA |= A_val;
602
603     seqB <= 1;
604     seqB |= B_val;
605     //Enmascaro los valores
606     seqA &= 0b00001111;
607     seqB &= 0b00001111;
608     //Compruebo si la secuencia coincide con las de un giro a derecha
609     if (seqA == 0b00001001 && seqB == 0b00000011)
610     {
611         stepper_irq_flag |= FLAG_IRQ_STEPPER_CONTINUE;
612         stepper_irq_flag |= FLAG_IRQ_STEPPER_DIR;
613         printf("He girado \n");
614         fflush(stdout);
615     }

```

```

616 //Compruebo si la secuencia coincide con las de un giro a izquierda
617 if (seqA == 0b00000011 && seqB == 0b00001001)
618 {
619     stepper_irq_flag |= FLAG_IRQ_STEPPER_CONTINUE;
620     stepper_irq_flag &= ~FLAG_IRQ_STEPPER_DIR;
621     printf("He girado \n");
622     fflush(stdout);
623 }
624 }

```

4.6.2.19 killRFID()

```
void killRFID ( )
```

Mata la maquina de estados del rfid.

Definition at line 412 of file fsm_rfid.c.

```

413 {
414     pthread_cancel(thread);
415     maquina_creada = 0;
416 }

```

4.6.2.20 launchRFID()

```
void launchRFID ( )
```

Funcion de creacion de la maquina de estados Si la maquina de estados ya esta creada no se crea otra.

Definition at line 287 of file fsm_rfid.c.

```

288 {
289     if (maquina_creada)
290     {
291         printf("La maquina de estados ya ha sido creada \n");
292         fflush(stdout);
293         return;
294     }
295     fsm_rfid = fsm_new(transition_table_rfid, NULL);
296     pthread_create(&thread, NULL, lp, fsm_rfid);
297     maquina_creada = 1;
298 }

```

4.6.2.21 LeerTarjeta()

```
void LeerTarjeta (
    fsm_t * fsm )
```

Lee la tarjeta, activa el flag de tarjeta valida Lee la tarjeta, almacena en una variable privada del fichero el UUID de la misma. Activa el flag de tarjeta valida.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Definition at line 198 of file fsm_rfid.c.

```

199 {
200     if (RC522_Check(UUID) == STATUS_OK)
201     {
202         printf("FOUND TAG! \n");
203         fflush(stdout);
204         flag_rfid |= FLAG_VALID_CARD;
205     }
206 }
```

4.6.2.22 lp()

```

void lp (
    void * userData )
```

Funcion de loop infinito de la maquina de estados.

Parameters

<i>userData</i>	Datos externos
-----------------	----------------

Definition at line 275 of file fsm_rfid.c.

```

276 {
277     fsm_t *data = (fsm_t *)userData;
278     while (1)
279     {
280         fsm_fire(data);
281     }
282 }
```

4.6.2.23 menu_display_stepper_plus()

```

void menu_display_stepper_plus (
    list_files_t * lista )
```

Refresco del LCD con los parametros de la lista Funcion de manejo del menu que se actualiza en funcion de lo que se haya operado en la funcion lista, para poder mostrar un menu rotativo bidireccional por el LCD. Anade en la primera linea del LCD el prompt de seleccion.

Parameters

<i>lista</i>	
--------------	--

Definition at line 425 of file fsm_rfid.c.

```

426 {
427     menu_lcd_display_clear();
428     if (stepper_irq_flag & FLAG_IRQ_STEPPER_DIR)
429         lista->current_file -= 2;
430
431     printf("Posicion %d \n", lista->current_file);
432     fflush(stdout);
433     char line1[20] = ">";
434     strcat(line1, get_next_file(lista));
435     int nextIndex = lista->current_file;
436     menu_lcd_display(line1, get_next_file(lista),
get_next_file(lista));
437     lista->current_file = nextIndex;
438 }
```

4.6.2.24 new_list_files()

```
list_files_t * new_list_files (
    int num_files )
```

Devuelve una estructura con memoria reservada y parametros por defecto.

Parameters

<i>num_files</i>	numero de elementos de la lista
------------------	---------------------------------

Returns

*list_files_t** puntero a la lista creada

Definition at line 536 of file fsm_rfid.c.

```

537 {
538     int i = 0;
539     list_files_t *lista = (list_files_t *)malloc(sizeof(
list_files_t));
540     lista->num_files = num_files;
541     lista->current_file = 0;
542     lista->select_file = 1;
543     lista->name_file = (char **)malloc(sizeof(char *) * num_files);
544     for (i = 0; i < num_files; i++)
545     {
546         lista->name_file[i] = (char *)malloc(sizeof(char *) * 20);
547     }
548     return lista;
549 }
```

4.6.2.25 select_mode()

```
void select_mode (
    int event )
```

Callback de boton del encoder Activa el flag de seleccion del encoder.

Parameters

<i>event</i>	evento que se ha producido
--------------	----------------------------

Definition at line 630 of file fsm_rfid.c.

```

631 {
632     lock(7);
633     stepper_irq_flag |= FLAG_IRQ_STEPPER_SELECT;
634     unlock(7);
635 }
```

4.6.2.26 TarjetaDisponible()

```

int TarjetaDisponible (
    fsm_t * fsm )
```

Condición de paso si hay una tarjeta insertada en el optoacoplador.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Returns

int Condición cumplida o no

Definition at line 126 of file fsm_rfid.c.

```

127 {
128     lock(0);
129     uint8_t tmp = (flag_rfid & FLAG_CARD_IN);
130     unlock(0);
131     return tmp;
132 }
```

4.6.2.27 TarjetaNoDisponible()

```

int TarjetaNoDisponible (
    fsm_t * fsm )
```

Condición de paso si no hay una tarjeta insertada en el optoacoplador.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Returns

int Condición cumplida o no

Definition at line 139 of file fsm_rfid.c.

```
140 {  
141     lock(0);  
142     uint8_t tmp = !(flag_rfid & FLAG_CARD_IN);  
143     unlock(0);  
144     return tmp;  
145 }
```

4.6.2.28 TarjetaNoValida()

```
int TarjetaNoValida (  
    fsm_t * fsm )
```

Condición de paso si la tarjeta no es RFID.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Returns

int Condición cumplida o no

Definition at line 165 of file fsm_rfid.c.

```
166 {  
167     lock(0);  
168     uint8_t tmp = !(flag_rfid & FLAG_VALID_CARD);  
169     unlock(0);  
170     return tmp;  
171 }
```

4.6.2.29 TarjetaValida()

```
int TarjetaValida (  
    fsm_t * fsm )
```

Condición de paso si la tarjeta es RFID.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Returns

int Condición cumplida o no

Definition at line 152 of file fsm_rfid.c.

```
153 {  
154     lock(0);  
155     uint8_t tmp = flag_rfid & FLAG_VALID_CARD;  
156     unlock(0);  
157     return tmp;  
158 }
```

4.6.2.30 UUID_2_int()

```
int UUID_2_int ( )
```

Conversor de Int Array a int del UUID.

Returns

int UUID en formato int

Definition at line 570 of file fsm_rfid.c.

```
571 {  
572     unsigned int id;  
573     int i = 0;  
574     for (i = 0; i < 8; i++)  
575     {  
576         id = (id << 8) | UUID[i];  
577     }  
578     return id;  
579 }
```

4.6.3 Variable Documentation**4.6.3.1 maquina_creada**

```
int maquina_creada = 0
```

Indica si la maquina de estados ha sido creada ya o no.

Definition at line 32 of file fsm_rfid.c.

4.6.3.2 seqA

```
int seqA
```

Definition at line 586 of file fsm_rfid.c.

4.6.3.3 seqB

```
int seqB
```

Definition at line 587 of file fsm_rfid.c.

4.6.3.4 transition_table_rfid

```
fsm_trans_t transition_table_rfid[]
```

Initial value:

```
= {  
    {WAIT_START, CompruebaComienzo, WAIT_CARD,  
      ComienzaSistema},  
    {WAIT_CARD, TarjetaDisponible, WAIT_CHECK,  
      LeerTarjeta},  
    {WAIT_CARD, TarjetaNoDisponible, WAIT_CARD,  
      EsperaTarjeta},  
    {WAIT_CHECK, TarjetaNoValida, WAIT_CARD,  
      DescartaTarjeta},  
    {WAIT_CHECK, TarjetaValida, WAIT_DATA,  
      BuscaTarjeta},  
    {WAIT_DATA, TarjetaNoExiste, WAIT_PLAY, ConfiguraTarjeta},  
    {WAIT_DATA, TarjetaExiste, WAIT_PLAY, ComienzaReproduccion},  
    {WAIT_PLAY, TarjetaDisponible, WAIT_PLAY,  
      CompruebaTarjeta},  
    {WAIT_PLAY, TarjetaNoDisponible, WAIT_START,  
      CancelaReproduccion},  
    {WAIT_PLAY, CompruebaFinalReproduccion,  
      WAIT_START, FinalizaReproduccion},  
    {-1, NULL, -1, NULL}}
```

Definition at line 50 of file fsm_rfid.c.

4.6.3.5 UUID

```
uint8_t UUID[16]
```

Variable privada con el UUID de la ultima tarjeta.

Definition at line 28 of file fsm_rfid.c.

4.7 fsm_rfid.h File Reference

```
#include <stdio.h>
#include <pthread.h>
#include <sched.h>
#include "fsm.h"
#include "defines.h"
#include "tipos.h"
#include "RC522.h"
```

Data Structures

- struct [list_files](#)

Typedefs

- typedef struct [list_files](#) [list_files_t](#)

Enumerations

- enum [flags_rfid](#) {
 [FLAG_SYSTEM_STARTn](#) = 0x01, [FLAG_CARD_IN](#) = 0x02, [FLAG_VALID_CARD](#) = 0x04, [FLAG_SYSTEM_END](#) = 0x08,
 [FLAG_CARD_EXIST](#) = 0x10 }
- enum [rfid_states](#) {
 [WAIT_START](#), [WAIT_PLAY](#), [WAIT_CARD](#), [WAIT_CHECK](#),
 [WAIT_CONFIG](#), [WAIT_DATA](#) }

Functions

- int [CompruebaComienzo](#) ([fsm_t](#) *fsm)
 Funcion de arranque del sistema devuelve siempre 1.
- int [CompruebaFinalReproduccion](#) ([fsm_t](#) *fsm)
 Condición de paso si Ha acabado la reproducción de la cancion.
- int [TarjetaDisponible](#) ([fsm_t](#) *fsm)
 Condición de paso si hay una tarjeta insertada en el optoacoplador.
- int [TarjetaNoDisponible](#) ([fsm_t](#) *fsm)
 Condición de paso si no hay una tarjeta insertada en el optoacoplador.
- int [TarjetaValida](#) ([fsm_t](#) *fsm)
 Condición de paso si la tarjeta es RFID.
- int [TarjetaNoValida](#) ([fsm_t](#) *fsm)
 Condición de paso si la tarjeta no es RFID.
- void [EsperaTarjeta](#) ([fsm_t](#) *fsm)
 Funcion de espera de la tarjeta mientras no haya una en el optoacoplador.
- void [DescartaTarjeta](#) ([fsm_t](#) *fsm)
 Altera el flag de tarjeta insertada.
- void [LeerTarjeta](#) ([fsm_t](#) *fsm)

Lee la tarjeta, activa el flag de tarjeta valida Lee la tarjeta, almacena en una variable privada del fichero el UUID de la misma. Activa el flag de tarjeta valida.

- void [ComienzaSistema](#) (fsm_t *fsm)

Funcion de Inicizalizacion del sistema.

- void [CancelaReproduccion](#) (fsm_t *fsm)

Activa el flag de inicio de stop de la maquina de estados de player.

- void [ComienzaReproduccion](#) (fsm_t *fsm)

Activa el flag de inicio de reproduccion de la maquina de estados de player.

- void [FinalizaReproduccion](#) (fsm_t *fsm)

Funcion de finalizacion de reproduccion.

- void [CompruebaTarjeta](#) (fsm_t *fsm)

Funcion de espera de comprueba tarjeta.

- void [launchRFID](#) ()

Funcion de creacion de la maquina de estados Si la maquina de estados ya esta creada no se crea otra.

Variables

- uint8_t [UUID](#) [16]

- fsm_t * [fsm_rfid](#)

Puntero fsm de la maquina de estados.

- pthread_t [thread](#)

Hilo de ejecucion de la maquina de estados.

4.7.1 Typedef Documentation

4.7.1.1 list_files_t

```
typedef struct list_files list_files_t
```

4.7.2 Enumeration Type Documentation

4.7.2.1 flags_rfid

```
enum flags_rfid
```

Enumerator

FLAG_SYSTEM_STARTn	Flaf de systema iniciado NO USADO.
FLAG_CARD_IN	Flag de tarjeta detectada optoacoplador.
FLAG_VALID_CARD	Flag tarjeta valida RFID.
FLAG_SYSTEM_END	Final del sistema.
FLAG_CARD_EXIST	Flag tarjeta Existe.

Definition at line 30 of file fsm_rfid.h.

```

30      {
31          FLAG_SYSTEM_STARTn = 0x01,
32          FLAG_CARD_IN = 0x02,
33          FLAG_VALID_CARD = 0x04,
34          FLAG_SYSTEM_END = 0x08,
35          FLAG_CARD_EXIST = 0x10
36 };

```

4.7.2.2 rfid_states

```
enum rfid_states
```

Enumerator

WAIT_START	
WAIT_PLAY	
WAIT_CARD	
WAIT_CHECK	
WAIT_CONFIG	
WAIT_DATA	

Definition at line 37 of file fsm_rfid.h.

```

37      {
38          WAIT_START,
39          WAIT_PLAY,
40          WAIT_CARD,
41          WAIT_CHECK,
42          WAIT_CONFIG,
43          WAIT_DATA
44 };

```

4.7.3 Function Documentation

4.7.3.1 CancelaReproduccion()

```
void CancelaReproduccion (
    fsm_t * fsm )
```

Activa el flag de inicio de stop de la maquina de estados de player.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Definition at line 225 of file fsm_rfid.c.


```

226 {
227     lock(1);
228     flags_player |= FLAG_END;
229     unlock(1);
230 }

```

4.7.3.2 ComienzaReproduccion()

```

void ComienzaReproduccion (
    fsm_t * fsm )

```

Activa el flag de inicio de reproduccion de la maquina de estados de player.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Definition at line 212 of file fsm_rfid.c.

```

213 {
214     menu_lcd_display("", "Playing: ", song_name, "");
215     lock(1);
216     flags_player = FLAG_START;
217     unlock(1);
218     printf("comienzas flag: %d", flags_player);
219 }

```

4.7.3.3 ComienzaSistema()

```

void ComienzaSistema (
    fsm_t * fsm )

```

Funcion de Inicizalizacion del sistema.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Definition at line 256 of file fsm_rfid.c.

```

257 {
258     printf("Arranca el RFID");
259     menu_lcd_display_clear(),
260     menu_lcd_display("PiMusicBox Player!", "", "", "");
261     fflush(stdout);
262     //Si no es la primera inicialización no ocurre nada aqui
263     if (RC522_Init() == STATUS_ERROR)
264         printf("ERROR! \n");
265     //Reset de todos los flags
266     lock(0);
267     flag_rfid = 0;
268     unlock(0);
269 }

```

4.7.3.4 CompruebaComienzo()

```
int CompruebaComienzo (  
    fsm_t * fsm )
```

Funcion de arranque del sistema devuelve siempre 1.

Parameters

<i>fsm</i>	
------------	--

Returns

int, siempre 1

Definition at line 103 of file fsm_rfid.c.

```
104 {  
105     return 1;  
106 }
```

4.7.3.5 CompruebaFinalReproduccion()

```
int CompruebaFinalReproduccion (  
    fsm_t * fsm )
```

Condición de paso si Ha acabado la reproducción de la cancion.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Returns

int Condición cumplida o no

Definition at line 113 of file fsm_rfid.c.

```
114 {  
115     lock(0);  
116     uint8_t tmp = (flag_rfid & FLAG_SYSTEM_END);  
117     unlock(0);  
118     return tmp;  
119 }
```

4.7.3.6 CompruebaTarjeta()

```
void CompruebaTarjeta (
    fsm_t * fsm )
```

Funcion de espera de comprueba tarjeta.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Definition at line 236 of file fsm_rfid.c.

```
237 {
238     return;
239 }
```

4.7.3.7 DescartaTarjeta()

```
void DescartaTarjeta (
    fsm_t * fsm )
```

Altera el flag de tarjeta insertada.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Definition at line 186 of file fsm_rfid.c.

```
187 {
188     lock(0);
189     flag_rfid &= ~FLAG_CARD_IN;
190     unlock(0);
191 }
```

4.7.3.8 EsperaTargeta()

```
void EsperaTargeta (
    fsm_t * fsm )
```

Funcion de espera de la tarjeta mientras no haya una en el optoacoplador.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Definition at line 177 of file fsm_rfid.c.

```
178 {
179     return;
180 }
```

4.7.3.9 FinalizaReproduccion()

```
void FinalizaReproduccion (
    fsm_t * fsm )
```

Funcion de finalizacion de reproduccion.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Definition at line 245 of file fsm_rfid.c.

```
246 {
247     lock(1);
248     flags_player |= FLAG_END;
249     unlock(1);
250 }
```

4.7.3.10 launchRFID()

```
void launchRFID ( )
```

Funcion de creacion de la maquina de estados Si la maquina de estados ya esta creada no se crea otra.

Definition at line 287 of file fsm_rfid.c.

```
288 {
289     if (maquina_creada)
290     {
291         printf("La maquina de estados ya ha sido creada \n");
292         fflush(stdout);
293         return;
294     }
295     fsm_rfid = fsm_new(transition_table_rfid, NULL);
296     pthread_create(&thread, NULL, lp, fsm_rfid);
297     maquina_creada = 1;
298 }
```

4.7.3.11 LeerTarjeta()

```
void LeerTarjeta (
    fsm_t * fsm )
```

Lee la tarjeta, activa el flag de tarjeta valida Lee la tarjeta, almacena en una variable privada del fichero el UUID de la misma. Activa el flag de tarjeta valida.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Definition at line 198 of file fsm_rfid.c.

```
199 {
200     if (RC522_Check(UUID) == STATUS_OK)
201     {
202         printf("FOUND TAG! \n");
203         fflush(stdout);
204         flag_rfid |= FLAG_VALID_CARD;
205     }
206 }
```

4.7.3.12 TarjetaDisponible()

```
int TarjetaDisponible (
    fsm_t * fsm )
```

Condición de paso si hay una tarjeta insertada en el optoacoplador.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Returns

int Condición cumplida o no

Definition at line 126 of file fsm_rfid.c.

```
127 {
128     lock(0);
129     uint8_t tmp = (flag_rfid & FLAG_CARD_IN);
130     unlock(0);
131     return tmp;
132 }
```

4.7.3.13 TarjetaNoDisponible()

```
int TarjetaNoDisponible (
    fsm_t * fsm )
```

Condición de paso si no hay una tarjeta insertada en el optoacoplador.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Returns

int Condición cumplida o no

Definition at line 139 of file fsm_rfid.c.

```
140 {  
141     lock(0);  
142     uint8_t tmp = !(flag_rfid & FLAG_CARD_IN);  
143     unlock(0);  
144     return tmp;  
145 }
```

4.7.3.14 TarjetaNoValida()

```
int TarjetaNoValida (  
    fsm_t * fsm )
```

Condición de paso si la tarjeta no es RFID.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Returns

int Condición cumplida o no

Definition at line 165 of file fsm_rfid.c.

```
166 {  
167     lock(0);  
168     uint8_t tmp = !(flag_rfid & FLAG_VALID_CARD);  
169     unlock(0);  
170     return tmp;  
171 }
```

4.7.3.15 TarjetaValida()

```
int TarjetaValida (  
    fsm_t * fsm )
```

Condición de paso si la tarjeta es RFID.

Parameters

<i>fsm</i>	Maquina de estados
------------	--------------------

Returns

int Condición cumplida o no

Definition at line 152 of file fsm_rfid.c.

```
153 {  
154     lock(0);  
155     uint8_t tmp = flag_rfid & FLAG_VALID_CARD;  
156     unlock(0);  
157     return tmp;  
158 }
```

4.7.4 Variable Documentation

4.7.4.1 fsm_rfid

```
fsm_t* fsm_rfid
```

Puntero fsm de la maquina de estados.

Definition at line 47 of file fsm_rfid.h.

4.7.4.2 thread

```
pthread_t thread
```

Hilo de ejecucion de la maquina de estados.

Definition at line 48 of file fsm_rfid.h.

4.7.4.3 UUID

```
uint8_t UUID[16]
```

Definition at line 22 of file fsm_rfid.h.

4.8 InterruptSM.c File Reference

```
#include "InterruptSM.h"
#include <stdio.h>
#include "bcm2835.h"
#include "poll.h"
```

Macros

- `#define THRESHOLD_HIGH 12`
Umbral para que el muestreo concluya nivel alto.
- `#define THRESHOLD_LOW 7`
Umbral para que el muestreo concluya nivel bajo.

Functions

- void `loop` (void *userData)
Bucle de interrupción.
- void `attachlsr` (uint8_t PIN, uint8_t ISREvent, void *handle, void *userData)
Crea la interrupcion Crea un hilo donde se lanza una funcion que analizara el estado del pin para detectar cambios en el.
- void `deletelsr` (uint8_t PIN)
Elimina la interrupcion del pin especificado.

4.8.1 Macro Definition Documentation

4.8.1.1 THRESHOLD_HIGH

```
#define THRESHOLD_HIGH 12
```

Umbral para que el muestreo concluya nivel alto.

Definition at line 12 of file InterruptSM.c.

4.8.1.2 THRESHOLD_LOW

```
#define THRESHOLD_LOW 7
```

Umbral para que el muestreo concluya nivel bajo.

Definition at line 13 of file InterruptSM.c.

4.8.2 Function Documentation

4.8.2.1 attachIsr()

```
void attachIsr (
    uint8_t PIN,
    uint8_t ISREvent,
    void * handdle,
    void * userData )
```

Crea la interrupcion Crea un hilo donde se lanza una funcion que analizara el estado del pin para detectar cambios en el.

Parameters

<i>PIN</i>	pin donde se crea la interrupcion
<i>ISREvent</i>	Tipo de evento que queremos detectar
<i>handdle</i>	(no used yet)
<i>userData</i>	funcion de callback

Definition at line 62 of file InterruptSM.c.

```
63 {
64     printf("Entro");
65     fflush(stdout);
66     if (!bcm2835_init())
67         return;
68     // Set RPI pin P1-15 to be an input
69     bcm2835_gpio_fsel(PIN, BCM2835_GPIO_FSEL_INPT);
70     // with a pullup
71     bcm2835_gpio_set_pud(PIN, BCM2835_GPIO_PUD_UP);
72     // And a low detect enable
73     pthread_t thread;
74     ISR_Typ_ *interrupt = (ISR_Typ_ *)malloc(sizeof(ISR_Typ_));
75
76     interrupt->callback = userData;
77     interrupt->event = ISREvent;
78     interrupt->pin = PIN;
79
80     printf("Entro");
81     fflush(stdout);
82     if (threads[PIN] != 0)
83     {
84         printf("Error, has been created yet! \n");
85         return;
86     }
87     pthread_create(&thread, NULL, loop, interrupt);
88     threads[PIN] = thread;
89 }
```

4.8.2.2 deletIsr()

```
void deleteIsr (
    uint8_t PIN )
```

Elimina la interrupcion del pin especificado.

Parameters

<i>PIN</i>	
------------	--

Definition at line 95 of file InterruptSM.c.

```

96 {
97     pthread_cancel(threads[PIN]);
98     threads[PIN] = 0;
99 }
```

4.8.2.3 loop()

```

void loop (
    void * userData )
```

Bucle de interrupción.

Parameters

<i>userData</i>	
-----------------	--

Definition at line 20 of file InterruptSM.c.

```

21 {
22     ISR_Typ_ *ptr = (ISR_Typ_ *)userData;
23     uint8_t pin = ptr->pin;
24     int suma = 0;
25     int contador = 0;
26     int last_val = 0;
27     while (1)
28     {
29         if (contador == 20)        //analizo en una ventana de 20 muestras
30         {
31             if (suma >= THRESHOLD_HIGH)        //si es un '1'
32             {
33                 if (last_val == 0 && (ptr->event == FALLIN_EDGE || ptr->
event == CHANGE))
34                 {
35                     ptr->callback(RISING_EDGE);
36                 }
37                 last_val = 1;
38             }
39             else if (suma <= THRESHOLD_LOW)        //si es un '0'
40             {
41                 if (last_val == 1 && (ptr->event == RISING_EDGE || ptr->
event == CHANGE))
42                 {
43                     ptr->callback(FALLIN_EDGE);        //llamo a la funcion del callback
44                 }
45                 last_val = 0;
46             }
47             suma = 0;
48             contador = 0;
49         }
50         suma += bcm2835_gpio_lev(pin);
51         contador++;
52     }
53 }
```

4.9 InterruptSM.h File Reference

```
#include <pthread.h>
#include <sched.h>
#include <stdint.h>
```

Data Structures

- struct [ISR_Typ](#)

Typedefs

- typedef void(* [call_back](#)) (int)
- typedef struct [ISR_Typ](#) [ISR_Typ_](#)

Enumerations

- enum [event](#) {
 [FALLIN_EDGE](#), [RISING_EDGE](#), [LOW_DETECT](#), [HIGH_DETECT](#),
 [CHANGE](#) }

Functions

- void [attachIsr](#) (uint8_t [PIN](#), uint8_t ISREvent, void *handle, void *userData)
 Crea la interrupcion Crea un hilo donde se lanza una funcion que analizara el estado del pin para detectar cambios en el.
- void [deletelsr](#) (uint8_t [PIN](#))
 Elimina la interrupcion del pin especificado.

Variables

- pthread_t [threads](#) [64]
 Hilos de las interrupciones.
- uint8_t [atachPin](#)

4.9.1 Typedef Documentation

4.9.1.1 call_back

```
typedef void(* call_back) (int)
```

Definition at line 16 of file InterruptSM.h.

4.9.1.2 ISR_Typ_

```
typedef struct ISR_Typ ISR_Typ_
```

4.9.2 Enumeration Type Documentation

4.9.2.1 event

```
enum event
```

Enumerator

FALLIN_EDGE	Flanco de bajada.
RISING_EDGE	Flanco de subida.
LOW_DETECT	Deteccion nivel alto.
HIGH_DETECT	Deteccion nivel bajo.
CHANGE	Cambio de nivel.

Definition at line 31 of file InterruptSM.h.

```
32 {
33     FALLIN_EDGE,
34     RISING_EDGE,
35     LOW_DETECT,
36     HIGH_DETECT,
37     CHANGE
38 };
```

4.9.3 Function Documentation

4.9.3.1 attachIsr()

```
void attachIsr (
    uint8_t PIN,
    uint8_t ISREvent,
    void * handdle,
    void * userData )
```

Crea la interrupcion Crea un hilo donde se lanza una funcion que analizara el estado del pin para detectar cambios en el.

Parameters

<i>PIN</i>	pin donde se crea la interrupcion
<i>ISREvent</i>	Tipo de evento que queremos detectar
<i>handdle</i>	(no used yet)
<i>userData</i>	funcion de callback

Definition at line 62 of file InterruptSM.c.

```

63 {
64     printf("Entro");
65     fflush(stdout);
66     if (!bcm2835_init())
67         return;
68     // Set RPI pin P1-15 to be an input
69     bcm2835_gpio_fsel(PIN, BCM2835_GPIO_FSEL_INPT);
70     // with a pullup
71     bcm2835_gpio_set_pud(PIN, BCM2835_GPIO_PUD_UP);
72     // And a low detect enable
73     pthread_t thread;
74     ISR_Typ_ *interrupt = (ISR_Typ_ *)malloc(sizeof(ISR_Typ_));
75
76     interrupt->callback = userData;
77     interrupt->event = ISREvent;
78     interrupt->pin = PIN;
79
80     printf("Entro");
81     fflush(stdout);
82     if (threads[PIN] != 0)
83     {
84         printf("Error, has been created yet! \n");
85         return;
86     }
87     pthread_create(&thread, NULL, loop, interrupt);
88     threads[PIN] = thread;
89 }

```

4.9.3.2 deletelsr()

```

void deleteIsr (
    uint8_t PIN )

```

Elimina la interrupcion del pin especificado.

Parameters

<i>PIN</i>	
------------	--

Definition at line 95 of file InterruptSM.c.

```

96 {
97     pthread_cancel(threads[PIN]);
98     threads[PIN] = 0;
99 }

```

4.9.4 Variable Documentation

4.9.4.1 attachPin

```
uint8_t attachPin
```

Definition at line 20 of file InterruptSM.h.

4.9.4.2 threads

```
pthread_t threads[64]
```

Hilos de las interrupciones.

Definition at line 19 of file InterruptSM.h.

4.10 mutex.c File Reference

```
#include <pthread.h>
```

Functions

- void [lock](#) (int key)
Bloquea el hilo actual.
- void [unlock](#) (int key)
desloquea el hilo actual

4.10.1 Function Documentation

4.10.1.1 lock()

```
void lock (  
    int key )
```

Bloquea el hilo actual.

Parameters

<i>key</i>	inidice del hilo a bloquear
------------	-----------------------------

Definition at line 16 of file mutex.c.

```
17 {  
18     pthread_mutex_lock (&piMutexes[key]) ;  
19 }
```

4.10.1.2 unlock()

```
void unlock (
    int key )
```

desloquea el hilo actual

Parameters

<i>key</i>	inidice del hilo a desbloquear
------------	--------------------------------

Definition at line 25 of file mutex.c.

```
26 {  
27     pthread_mutex_unlock (&piMutexes[key]) ;  
28 }
```

4.11 mutex.h File Reference

Functions

- void **lock** (int key)
Bloquea el hilo actual.
- void **unlock** (int key)
desloquea el hilo actual

4.11.1 Function Documentation

4.11.1.1 lock()

```
void lock (  
           int key )
```

Bloquea el hilo actual.

Parameters

<i>key</i>	inidice del hilo a bloquear
------------	-----------------------------

Definition at line 16 of file mutex.c.

```
17 {  
18     pthread_mutex_lock (&piMutexes[key]) ;  
19 }
```

4.11.1.2 unlock()

```
void unlock (  
            int key )
```

desloquea el hilo actual

Parameters

<i>key</i>	inidice del hilo a desbloquear
------------	--------------------------------

Definition at line 25 of file mutex.c.

```

26 {
27     pthread_mutex_unlock(&piMutexes[key]) ;
28 }
```

4.12 piMusicBox_2.c File Reference

```

#include "fsm.h"
#include "player.h"
#include <stdio.h>
#include <pthread.h>
#include <sched.h>
#include "RC522.h"
#include "fsm_rfid.h"
#include "piMusicBox_2.h"
#include "InterruptSM.h"
#include "mutex.h"
#include "menu_lcd.h"
#include "dbcontroller.h"
#include <time.h>
```

Functions

- void [callback](#) (int [event](#))
Callback llamado cuando se produce una interrupción.
- int [main](#) (void)

Variables

- int [k](#) = 0

4.12.1 Function Documentation**4.12.1.1 callback()**

```

void callback (
    int event )
```

Callback llamado cuando se produce una interrupción.

Parameters

<i>event</i>	tipo de evento que lo ha causado
--------------	----------------------------------

Definition at line 36 of file piMusicBox_2.c.

```

37 {
38     if (k == 0)
39     {
40         k++;
41         return;
42     }
43     if (event == FALLIN_EDGE)
44     {
45         printf("FALLING event");
46         fflush(stdout);
47         lock(1);
48         flag_rfid |= FLAG_CARD_IN;
49         unlock(1);
50     }
51     else if (event == RISING_EDGE)
52     {
53         printf("RISING event");
54         fflush(stdout);
55         lock(1);
56         flag_rfid &= ~FLAG_CARD_IN;
57         unlock(1);
58     }
59 }
```

4.12.1.2 main()

```

int main (
    void )
```

Definition at line 19 of file piMusicBox_2.c.

```

20 {
21
22     attachIsr(18, CHANGE, NULL, callback);    //creo interrupcion para el
        optoacoplador
23
24     menu_lcd_init();                          //inicio la pantalla
25     menu_lcd_display("PiBox RFID player!", "", "", "");
26     launchPlayer();                          //lanzo maquina de estados del player
27     launchRFID();                            //lanzo maquina de estados del rfid
28     while (1)
29         ;
30 }
```

4.12.2 Variable Documentation

4.12.2.1 k

```
int k = 0
```

Definition at line 15 of file piMusicBox_2.c.

4.13 piMusicBox_2.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <portaudio.h>
#include "defines.h"
#include "tipos.h"
#include "bcm2835.h"
#include "tone.h"
```

4.14 player.c File Reference

```
#include "player.h"
#include "mutex.h"
#include <string.h>
```

Functions

- void [output](#) (struct mad_pcm *pcm, [BUFFERS_T](#) *buffer, uint8_t cbuff)
Carga los frame en el buffer Funcion auxiliar que carga los datos del frame en buffer que corresponda en el momento.
- void [Iniciliza_player](#) (fsm_t *userData)
Inicia el las estructuras necesarias Inicia los estructuras de decodificacion y reproduccion necesarias. Tambien carga el primer frame decodificado+ en el buffer 1.
- void [carga_bff1](#) (fsm_t *userData)
Carga el buffer 1 Decodifica y sintetisa un frame y lo almacena en el buffer 1.
- void [carga_bff2](#) (fsm_t *userData)
Carga el buffer 2 Decodifica y sintetisa un frame y lo almacena en el buffer 2.
- [BUFFERS_T](#) * [new_buffer](#) (void)
Crea una estructura de buffers.
- void [Final_Melodia](#) (fsm_t *userData)
Finaliza la melodia Limpia la estructuras utilizadas y cierra los ficheros.
- void [func](#) (void *data)
funcion de loop del fsm
- void [launchPlayer](#) ()
inicia el reproductor de audio Crea la maquina de estados y la lanza en un hilo de alta prioridad para garantizar tiempo real

Variables

- struct mad_stream [mad_stream](#)
Estructura de stream de audio.
- struct mad_frame [mad_frame](#)
Estructura de informacion del frame decodificado.
- struct mad_synth [mad_synth](#)
Estructura de sintesis de frame.

- PaStreamParameters [outputParameters](#)
Estructura de parametros del driver de audio.
- PaStream * [stream](#)
Estructura de Stream de reproduccion.
- PaError [err](#)
Variable de error.
- FILE * [fp](#)
Puntero a fichero. Contendrá el fichero .mp3.
- [fsm_trans_t transition_table_player](#) []

4.14.1 Function Documentation

4.14.1.1 carga_bff1()

```
void carga_bff1 (
    fsm_t * userData )
```

Carga el buffer 1 Decodifica y sintetisa un frame y lo almacena en el buffer 1.

Parameters

userData	Estructura de datos de usuario
--------------------------	--------------------------------

Definition at line 218 of file player.c.

```
219 {
220     fsm_audio_controller_t *data = (fsm_audio_controller_t *)
    userData;
221     flags_player &= ~FLAG_BFF1_END;
222     mad_frame_decode(&mad_frame, &mad_stream);
223     // Synthesize PCM data of frame
224     mad_synth_frame(&mad_synth, &mad_frame);
225     output(&mad_synth.pcm, data->buffer, 0);
226 }
```

4.14.1.2 carga_bff2()

```
void carga_bff2 (
    fsm_t * userData )
```

Carga el buffer 2 Decodifica y sintetisa un frame y lo almacena en el buffer 2.

Parameters

userData	Estructura de datos de usuario
--------------------------	--------------------------------

Definition at line 203 of file player.c.

```

204 {
205
206     fsm_audio_controller_t *data = (fsm_audio_controller_t *)
        userData;
207     flags_player &= ~FLAG_BFF2_END;
208     mad_frame_decode(&mad_frame, &mad_stream);
209     // Synthesize PCM data of frame
210     mad_synth_frame(&mad_synth, &mad_frame);
211     output(&mad_synth.pcm, data->buffer, 1);
212 }
```

4.14.1.3 Final_Melodia()

```

void Final_Melodia (
    fsm_t * userData )
```

Finaliza la melodía Limpia la estructuras utilizadas y cierra los ficheros.

Parameters

<i>userData</i>	Estructura de datos de usuario
-----------------	--------------------------------

Definition at line 234 of file player.c.

```

235 {
236     printf("Final de la melodía \n");
237     flags_player = 0;
238     Pa_StopStream(stream);
239     fclose(fp);
240
241     // Free MAD structs
242     mad_synth_finish(&mad_synth);
243     mad_frame_finish(&mad_frame);
244     mad_stream_finish(&mad_stream);
245 }
```

4.14.1.4 func()

```

void func (
    void * data )
```

funcion de loop del fsm

Parameters

<i>data</i>	
-------------	--

Definition at line 408 of file player.c.

```

409 {
```

```

410     fsm_audio_controller_t *sFsm = (fsm_audio_controller_t *)
data;
411     while (1)
412     {
413         fsm_fire(sFsm->fsm);
414     }
415 }

```

4.14.1.5 Iniciliza_player()

```

void Iniciliza_player (
    fsm_t * userData )

```

Inicia el las estrucuturas necesarias Inicia los estructuras de decodificacion y reproduccion necesarias. Tambien carga el primer frame decodificado+ en el buffer 1.

Parameters

<i>userData</i>	Estructura de datos de usuario
-----------------	--------------------------------

Definition at line 116 of file player.c.

```

117 {
118
119     fsm_audio_controller_t *data = (fsm_audio_controller_t *)
userData; //Cargo datos de usuario
120     data->buffer = new_buffer(); //Creo el la
estructura de buffers
121     flags_player &= ~FLAG_START;
122
123     //Cargo el fichero seleccionado en memoria
124     char filename[64] = "./musica/"; //ruta parcial dle fichero
125     strcat(filename, song_name);
126     printf("full path: %s \n", filename);
127     free(song_name);
128
129     fp = fopen(filename, "r");
130     int fd = fileno(fp); //obtengo puntero
131
132     struct stat metadata; //estructua de informacion de fichero
133
134     //compruebo que se ha cargado correctamente
135     if (fstat(fd, &metadata) >= 0)
136     {
137         printf("File size %d bytes\n", (int)metadata.st_size);
138     }
139     else
140     {
141         printf("Failed to stat %s\n", filename);
142         fclose(fp);
143         return;
144     }
145     //mapeo el fichero
146     const unsigned char *input_stream = mmap(0, metadata.st_size, PROT_READ, MAP_SHARED, fd, 0);
147     mad_stream_buffer(&mad_stream, input_stream, metadata.st_size);
148
149     //CARGAMOS - PROCESAMOS BFF1
150     mad_frame_decode(&mad_frame, &mad_stream);
151
152     //sisntesis de frame
153     mad_synth_frame(&mad_synth, &mad_frame);
154     output(&mad_synth.pcm, data->buffer, 0);
155     flags_player |= FLAG_BFF2_END;
156
157     //Iniciamos el driver de audio
158     err = Pa_Initialize();
159     if (err != paNoError)
160         goto error;
161     //Cargmaos los dispositivos de reproduccion disponibles del sistema. nos quedamos con el por defecto

```

```

162     outputParameters.device = Pa_GetDefaultOutputDevice(); /* default output device */
163     if (outputParameters.device == paNoDevice)
164     {
165         fprintf(stderr, "Error: No default output device.\n");
166         goto error;
167     }
168
169     //Parametros de reproduccion
170     outputParameters.channelCount = 2;          // stereo
171     outputParameters.sampleFormat = paInt32;    // 32 bit (tipo de muestras )
172     outputParameters.suggestedLatency = Pa_GetDeviceInfo(
outputParameters.device)->defaultLowOutputLatency;
173     outputParameters.hostApiSpecificStreamInfo = NULL;
174
175     //Abro el stream de repducción
176     Pa_OpenStream(
177         &stream,
178         NULL, // no input
179         &outputParameters,
180         SAMPLE_RATE, //velocidad de muestreo
181         FRAMES_PER_BUFFER, //Tamaño del frame, por defecto 1152 (especificacion mpeg layer
3)
182         paClipOff, //sin solapamiento
183         patestCallback, //funcion de callback
184         data->buffer);
185
186     //iniciamos el stream de audio
187     Pa_StartStream(stream);
188     return;
189
190 error:
191     Pa_Terminate();
192     fprintf(stderr, "An error ocured while using the portaudio stream\n");
193     fprintf(stderr, "Error number: %d\n", err);
194     fprintf(stderr, "Error message: %s\n", Pa_GetErrorText(err));
195     return;
196 }

```

4.14.1.6 launchPlayer()

```
void launchPlayer ( )
```

inicia el reproductor de audio Crea la maquina de estados y la lanza en un hilo de alta prioridad para garantizar tiempo real

Definition at line 380 of file player.c.

```

381 {
382     fsm_audio_controller_t *sFsm = (fsm_audio_controller_t *)
malloc(sizeof(fsm_audio_controller_t));
383     sFsm->fsm = fsm_new(transition_table_player, NULL);
384
385     pthread_attr_t tattr;
386     pthread_t thread;
387     int newprio = 99;
388     struct sched_param param;
389     /* initialized with default attributes */
390     pthread_attr_init(&tattr);
391
392     /* safe to get existing scheduling param */
393     pthread_attr_getschedparam(&tattr, &param);
394
395     /* set the priority; others are unchanged */
396     param.sched_priority = newprio;
397
398     /* setting the new scheduling param */
399     pthread_attr_setschedparam(&tattr, &param);
400     pthread_create(&thread, &tattr, func, sFsm);
401 }

```

4.14.1.7 new_buffer()

```
BUFFERS_T * new_buffer (
    void )
```

Crea una estructura de buffers.

Returns

BUFFERS_T* Puntero a la estructura creada

Definition at line 347 of file player.c.

```
348 {
349     BUFFERS_T *new = (BUFFERS_T *)malloc(sizeof(BUFFERS_T));
350     new->buffl_l = (int *)malloc(sizeof(int) * FRAMES_PER_BUFFER);
351     new->buffl_r = (int *)malloc(sizeof(int) * FRAMES_PER_BUFFER);
352     new->buff2_l = (int *)malloc(sizeof(int) * FRAMES_PER_BUFFER);
353     new->buff2_r = (int *)malloc(sizeof(int) * FRAMES_PER_BUFFER);
354     new->currentBuffer = 1;
355     new->lengthBuffer = FRAMES_PER_BUFFER;
356     new->sampleReaded = 0;
357     new->flags = 0;
358     return new;
359 }
```

4.14.1.8 output()

```
void output (
    struct mad_pcm * pcm,
    BUFFERS_T * buffer,
    uint8_t cbuff )
```

Carga los frame en el buffer Funcion auxiliar que carga los datos del frame en buffer que corresponda en el momento.

Parameters

<i>pcm</i>	Estructura de datos con mas muestras pcm del frame ya sintetizadas
<i>buffer</i>	Estructura del buffer de reproducción
<i>cbuff</i>	indice del buffer actual en el que hay que escribir

Definition at line 255 of file player.c.

```
256 {
257     register int nsamples = pcm->length;
258     mad_fixed_t const *left_ch = pcm->samples[0], *right_ch = pcm->samples[1]; //longitud del frame
259     int i = 0; //punteros a buffer pcm
260     //carga los buffers correspondientes a los dos canales
261     while (nsamples--)
262     {
263         signed int sample;
264         sample = *left_ch++;
265         if (cbuff == 0)
266             *(buffer->buffl_l + i) = sample;
267         else
268             *(buffer->buff2_l + i) = sample;
```



```
269
270     sample = *right_ch++;
271     if (cbuff == 0)
272         *(buffer->buff1_r + i) = sample;
273     else
274         *(buffer->buff2_r + i) = sample;
275
276     i++;
277 }
278 }
```

4.14.2 Variable Documentation

4.14.2.1 err

```
PaError err
```

Variable de error.

Definition at line 37 of file player.c.

4.14.2.2 fp

```
FILE* fp
```

Puntero a fichero. Contendrá el fichero .mp3.

Definition at line 38 of file player.c.

4.14.2.3 mad_frame

```
struct mad_frame mad_frame
```

Estructura de informacion del frame decodificado.

Definition at line 32 of file player.c.

4.14.2.4 mad_stream

```
struct mad_stream mad_stream
```

Estructura de stream de audio.

Definition at line 31 of file player.c.

4.14.2.5 mad_synth

```
struct mad_synth mad_synth
```

Estructura de sintesis de frame.

Definition at line 33 of file player.c.

4.14.2.6 outputParameters

```
PaStreamParameters outputParameters
```

Estructura de parametros del driver de audio.

Definition at line 35 of file player.c.

4.14.2.7 stream

```
PaStream* stream
```

Estructura de Stream de reproduccion.

Definition at line 36 of file player.c.

4.14.2.8 transition_table_player

```
fsm_trans_t transition_table_player[]
```

Initial value:

```
= {  
    {WAIT_BEGIN, CompruebaPlayerStart, WAIT_BFF1,  
      Iniciliza_player},  
    {WAIT_BFF1, CompruebaFinal, WAIT_BEGIN, Final_Melodia},  
    {WAIT_BFF1, Comprueba_fin_bff2, WAIT_BFF2, carga_bff2},  
    {WAIT_BFF2, CompruebaFinal, WAIT_BEGIN, Final_Melodia},  
    {WAIT_BFF2, Comprueba_fin_bff1, WAIT_BFF1, carga_bff1},  
    {-1, NULL, -1, NULL}}
```

Definition at line 41 of file player.c.

4.15 player.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <pthread.h>
#include <sched.h>
#include "mad.h"
#include "portaudio.h"
#include "tipos.h"
#include "fsm.h"
```

Data Structures

- struct [BUFFERS](#)
Estructura que almacena los buffers del Ping Pong.
- struct [fsm_audio_controller](#)
Estructura de la maquina de estados player.

Macros

- #define [SAMPLE_RATE](#) 44100
- #define [FRAMES_PER_BUFFER](#) 1152

Typedefs

- typedef struct [BUFFERS](#) [BUFFERS_T](#)
Estructura que almacena los buffers del Ping Pong.
- typedef struct [fsm_audio_controller](#) [fsm_audio_controller_t](#)
Estructura de la maquina de estados player.

Enumerations

- enum [fsm_states](#) { [WAIT_BEGIN](#), [WAIT_BFF1](#), [WAIT_BFF2](#) }

Functions

- void [launchPlayer](#) ()
inicia el reproductor de audio Crea la maquina de estados y la lanza en un hilo de alta prioridad para garantizar tiempo real

4.15.1 Macro Definition Documentation

4.15.1.1 FRAMES_PER_BUFFER

```
#define FRAMES_PER_BUFFER 1152
```

Definition at line 26 of file player.h.

4.15.1.2 SAMPLE_RATE

```
#define SAMPLE_RATE 44100
```

Definition at line 25 of file player.h.

4.15.2 Typedef Documentation

4.15.2.1 BUFFERS_T

```
typedef struct BUFFERS BUFFERS_T
```

Estructura que almacena los buffers del Ping Pong.

4.15.2.2 fsm_audio_controller_t

```
typedef struct fsm_audio_controller fsm_audio_controller_t
```

Estructura de la maquina de estados player.

4.15.3 Enumeration Type Documentation

4.15.3.1 fsm_states

```
enum fsm_states
```

Enumerator

WAIT_BEGIN	
WAIT_BFF1	
WAIT_BFF2	

Definition at line 29 of file player.h.

```

30 {
31     WAIT_BEGIN,
32     WAIT_BFF1,
33     WAIT_BFF2
34 };

```

4.15.4 Function Documentation

4.15.4.1 launchPlayer()

```
void launchPlayer ( )
```

inicia el reproductor de audio Crea la maquina de estados y la lanza en un hilo de alta prioridad para garantizar tiempo real

Definition at line 380 of file player.c.

```

381 {
382     fsm_audio_controller_t *sFsm = (fsm_audio_controller_t *)
    malloc(sizeof(fsm_audio_controller_t));
383     sFsm->fsm = fsm_new(transition_table_player, NULL);
384
385     pthread_attr_t tattr;
386     pthread_t thread;
387     int newprio = 99;
388     struct sched_param param;
389     /* initialized with default attributes */
390     pthread_attr_init(&tattr);
391
392     /* safe to get existing scheduling param */
393     pthread_attr_getschedparam(&tattr, &param);
394
395     /* set the priority; others are unchanged */
396     param.sched_priority = newprio;
397
398     /* setting the new scheduling param */
399     pthread_attr_setschedparam(&tattr, &param);
400     pthread_create(&thread, &tattr, func, sFsm);
401 }

```

4.16 tipos.h File Reference

```
#include <stdint.h>
```

Data Structures

- struct [TipoSistema](#)

Macros

- `#define FLAG_PLAYER_START 0x01`
- `#define FLAG_PLAYER_STOP 0x02`
- `#define FLAG_PLAYER_END 0x04`
- `#define FLAG_NOTA_TIMEOUT 0x08`
- `#define FLAG_QUIT 0x10`
- `#define FLAG_BFF1_END 0x01`
Flag de fin de buffer1.
- `#define FLAG_BFF2_END 0x02`
Flag de fin de buffer2.
- `#define FLAG_END 0x04`
Flag fin de cancion.
- `#define FLAG_START 0x08`
Flag de inicio de player.
- `#define FLAG_IRQ_STEPPER_CONTINUE 0x01`
Flag de movimiento detectado.
- `#define FLAG_IRQ_STEPPER_SELECT 0x02`
Flag de pulsacion de seleccion detectado.
- `#define FLAG_IRQ_STEPPER_DIR 0x04`
Flag de direccion de movimiento 0-> izq 1-> der.

Variables

- `volatile uint8_t flag_fsm`
flag de la maquina
- `volatile uint8_t flags_player`
flags del player
- `volatile uint8_t flag_rfid`
flags del rfid
- `volatile uint8_t stepper_irq_flag`
flags del encoder
- `volatile char * song_name`
numero del fichero
- `volatile int num_file`

4.16.1 Macro Definition Documentation

4.16.1.1 FLAG_BFF1_END

```
#define FLAG_BFF1_END 0x01
```

Flag de fin de buffer1.

Definition at line 29 of file tipos.h.

4.16.1.2 FLAG_BFF2_END

```
#define FLAG_BFF2_END 0x02
```

Flag de fin de buffer2.

Definition at line 30 of file tipos.h.

4.16.1.3 FLAG_END

```
#define FLAG_END 0x04
```

Flag fin de cancion.

Definition at line 31 of file tipos.h.

4.16.1.4 FLAG_IRQ_STEPPER_CONTINUE

```
#define FLAG_IRQ_STEPPER_CONTINUE 0x01
```

Flag de movimiento detectado.

Definition at line 35 of file tipos.h.

4.16.1.5 FLAG_IRQ_STEPPER_DIR

```
#define FLAG_IRQ_STEPPER_DIR 0x04
```

Flag de direccion de movimiento 0-> izq 1-> der.

Definition at line 37 of file tipos.h.

4.16.1.6 FLAG_IRQ_STEPPER_SELECT

```
#define FLAG_IRQ_STEPPER_SELECT 0x02
```

Flag de pulsacion de seleccion detectado.

Definition at line 36 of file tipos.h.

4.16.1.7 FLAG_NOTA_TIMEOUT

```
#define FLAG_NOTA_TIMEOUT 0x08
```

Definition at line 25 of file tipos.h.

4.16.1.8 FLAG_PLAYER_END

```
#define FLAG_PLAYER_END 0x04
```

Definition at line 24 of file tipos.h.

4.16.1.9 FLAG_PLAYER_START

```
#define FLAG_PLAYER_START 0x01
```

Definition at line 22 of file tipos.h.

4.16.1.10 FLAG_PLAYER_STOP

```
#define FLAG_PLAYER_STOP 0x02
```

Definition at line 23 of file tipos.h.

4.16.1.11 FLAG_QUIT

```
#define FLAG_QUIT 0x10
```

Definition at line 26 of file tipos.h.

4.16.1.12 FLAG_START

```
#define FLAG_START 0x08
```

Flag de inicio de player.

Definition at line 32 of file tipos.h.

4.16.2 Variable Documentation

4.16.2.1 flag_fsm

```
volatile uint8_t flag_fsm
```

flag de la maquina

Definition at line 39 of file tipos.h.

4.16.2.2 flag_rfid

```
volatile uint8_t flag_rfid
```

flags del rfid

Definition at line 41 of file tipos.h.

4.16.2.3 flags_player

```
volatile uint8_t flags_player
```

flags del player

Definition at line 40 of file tipos.h.

4.16.2.4 num_file

```
volatile int num_file
```

Definition at line 44 of file tipos.h.

4.16.2.5 song_name

```
volatile char* song_name
```

numero del fichero

Definition at line 43 of file tipos.h.

4.16.2.6 stepper_irq_flag

```
volatile uint8_t stepper_irq_flag
```

flags del encoder

Definition at line 42 of file tipos.h.

4.17 tone.c File Reference

```
#include "tone.h"
```

Functions

- void [toggle](#) ()
- void [tone_init](#) (uint8_t P)
- void [tone_write](#) (uint32_t freq)
- void [tone_stop](#) (void)

Variables

- volatile uint8_t [PIN](#)

4.17.1 Function Documentation

4.17.1.1 toggle()

```
void toggle ( )
```

Definition at line 93 of file tone.c.

```
94 {  
95     uint32_t halfPeriod;  
96     uint64_t tmp;  
97     while (1)  
98     {  
99         if (pin_freq == 0)  
100             bcm2835_delay(1);  
101         else  
102         {  
103             halfPeriod = 500000 / pin_freq;  
104             tmp = bcm2835_st_read();  
105             bcm2835_gpio_write(PIN, HIGH);  
106             bcm2835_delayMicroseconds(halfPeriod);  
107             bcm2835_gpio_write(PIN, LOW);  
108             bcm2835_delayMicroseconds(halfPeriod);  
109         }  
110     }  
111 }
```

4.17.1.2 tone_init()

```
void tone_init (
    uint8_t P )
```

Definition at line 30 of file tone.c.

```
31 {
32     if (P > 63)
33     {
34         printf("ERROR!. Pin %d out of bounds!", PIN);
35         return;
36     }
37     PIN = P;
38     if (pwm_thread != 0)
39     {
40         printf("Can not create Tone!., had it been created yet?");
41         return;
42     }
43     pthread_attr_t tattr;
44     int ret;
45     int newprio = 50;
46     struct sched_param param;
47     /* initialized with default attributes */
48     ret = pthread_attr_init(&tattr);
49
50     /* safe to get existing scheduling param */
51     ret = pthread_attr_getschedparam(&tattr, &param);
52
53     /* set the priority; others are unchanged */
54     param.sched_priority = newprio;
55
56     /* setting the new scheduling param */
57     ret = pthread_attr_setschedparam(&tattr, &param);
58
59     bcm2835_gpio_fsel(PIN, BCM2835_GPIO_FSEL_OUTP);
60     printf("\n PIN: %d \n", PIN);
61     pthread_create(&pwm_thread, &tattr, toggle, NULL);
62 }
```

4.17.1.3 tone_stop()

```
void tone_stop (
    void )
```

Definition at line 78 of file tone.c.

```
79 {
80     if (pwm_thread != 0)
81     {
82         pthread_cancel(pwm_thread);
83         pthread_join(pwm_thread, NULL);
84         bcm2835_gpio_write(PIN, LOW);
85     }
86 }
```

4.17.1.4 tone_write()

```
void tone_write (
    uint32_t freq )
```

Definition at line 68 of file tone.c.

```
69 {
70     if (freq < 0)
71         freq = 0;
72     else if (freq > 5000) // Max 5KHz
73         freq = 5000;
74
75     pin_freq = freq;
76 }
```

4.17.2 Variable Documentation

4.17.2.1 PIN

```
volatile uint8_t PIN
```

Definition at line 24 of file tone.c.

4.18 tone.h File Reference

```
#include <stdio.h>
#include <pthread.h>
#include <sched.h>
#include "bcm2835.h"
```

Functions

- void [tone_init](#) (uint8_t P)
- void [tone_write](#) (uint32_t freq)
- void [tone_stop](#) (void)

4.18.1 Function Documentation

4.18.1.1 tone_init()

```
void tone_init (
    uint8_t P )
```

Definition at line 30 of file tone.c.

```
31 {
32     if (P > 63)
33     {
34         printf("ERROR!. Pin %d out of bounds!", PIN);
35         return;
36     }
37     PIN = P;
38     if (pwm_thread != 0)
39     {
40         printf("Can not create Tone!., had it been created yet?");
41         return;
42     }
43     pthread_attr_t tattr;
44     int ret;
45     int newprio = 50;
46     struct sched_param param;
47     /* initialized with default attributes */
48     ret = pthread_attr_init(&tattr);
49
50     /* safe to get existing scheduling param */
51     ret = pthread_attr_getschedparam(&tattr, &param);
52
53     /* set the priority; others are unchanged */
54     param.sched_priority = newprio;
55
56     /* setting the new scheduling param */
57     ret = pthread_attr_setschedparam(&tattr, &param);
58
59     bcm2835_gpio_fsel(PIN, BCM2835_GPIO_FSEL_OUTP);
60     printf("\n PIN: %d \n", PIN);
61     pthread_create(&pwm_thread, &tattr, toggle, NULL);
62 }
```

4.18.1.2 tone_stop()

```
void tone_stop (
    void )
```

Definition at line 78 of file tone.c.

```
79 {
80     if (pwm_thread != 0)
81     {
82         pthread_cancel(pwm_thread);
83         pthread_join(pwm_thread, NULL);
84         bcm2835_gpio_write(PIN, LOW);
85     }
86 }
```

4.18.1.3 tone_write()

```
void tone_write (
    uint32_t freq )
```

Definition at line 68 of file tone.c.

```
69 {
70     if (freq < 0)
71         freq = 0;
72     else if (freq > 5000) // Max 5KHz
73         freq = 5000;
74
75     pin_freq = freq;
76 }
```


Index

atachPin
 InterruptSM.h, [65](#)

attachIsr
 InterruptSM.c, [61](#)
 InterruptSM.h, [64](#)

BUFFERS_T
 player.h, [80](#)

BUFFERS, [5](#)
 buff1_l, [6](#)
 buff1_r, [6](#)
 buff2_l, [6](#)
 buff2_r, [6](#)
 currentBuffer, [6](#)
 flags, [6](#)
 lengthBuffer, [7](#)
 sampleReaded, [7](#)

bcm_spi
 defines.h, [25](#)

buff1_l
 BUFFERS, [6](#)

buff1_r
 BUFFERS, [6](#)

buff2_l
 BUFFERS, [6](#)

buff2_r
 BUFFERS, [6](#)

buffer
 fsm_audio_controller, [9](#)

BuscaTarjeta
 fsm_rfid.c, [32](#)

CANCION_ACABADA
 defines.h, [25](#)

CANCION_NOACABADA
 defines.h, [26](#)

call_back
 InterruptSM.h, [63](#)

callback
 ISR_Typ, [11](#)
 piMusicBox_2.c, [69](#)

CancelaReproduccion
 fsm_rfid.c, [33](#)
 fsm_rfid.h, [52](#)

carga_bff1
 player.c, [72](#)

carga_bff2
 player.c, [72](#)

clean_list_files
 fsm_rfid.c, [33](#)

ComienzaReproduccion
 fsm_rfid.c, [34](#)
 fsm_rfid.h, [53](#)

ComienzaSistema
 fsm_rfid.c, [34](#)
 fsm_rfid.h, [53](#)

CompruebaComienzo
 fsm_rfid.c, [35](#)
 fsm_rfid.h, [54](#)

CompruebaFinalReproduccion
 fsm_rfid.c, [35](#)
 fsm_rfid.h, [54](#)

CompruebaTarjeta
 fsm_rfid.c, [36](#)
 fsm_rfid.h, [54](#)

ConfiguraTarjeta
 fsm_rfid.c, [36](#)

ConfiguracionCorrecta
 fsm_rfid.c, [36](#)

current_file
 list_files, [12](#)

current_state
 fsm_, [8](#)

currentBuffer
 BUFFERS, [6](#)

DB_NAME
 fsm_rfid.c, [32](#)

db_check
 dbcontroller.c, [17](#)
 dbcontroller.h, [22](#)

db_close
 dbcontroller.c, [17](#)
 dbcontroller.h, [23](#)

db_create_tables
 dbcontroller.c, [18](#)

db_free_song_name
 dbcontroller.c, [19](#)

db_get_song_name
 dbcontroller.c, [19](#)
 dbcontroller.h, [23](#)

db_insert
 dbcontroller.c, [20](#)
 dbcontroller.h, [24](#)

db_load
 dbcontroller.c, [21](#)
 dbcontroller.h, [25](#)

dbcontroller.c, [15](#)
 db_check, [17](#)
 db_close, [17](#)

- db_create_tables, 18
- db_free_song_name, 19
- db_get_song_name, 19
- db_insert, 20
- db_load, 21
- SQL_QUERY_CHECK_DB, 16
- SQL_QUERY_CREATE_DB, 16
- SQL_QUERY_GET_NUMBER_OF_ELEMENTS, 16
- SQL_QUERY_INSERT_DB, 16
- SQL_QUERY_MAX_SIZE, 16
- SQL_QUERY_SELECT_DB, 16
- SQL_TABLE_NAME, 17
- dbcontroller.h, 21
 - db_check, 22
 - db_close, 23
 - db_get_song_name, 23
 - db_insert, 24
 - db_load, 25
 - SQL_RESOLUTION, 22
- defines.h, 25
 - bcm_spi, 25
 - CANCION_ACABADA, 25
 - CANCION_NOACABADA, 26
 - PIN_PWM, 26
 - use_bcm, 26
- deletelsr
 - InterruptSM.c, 61
 - InterruptSM.h, 65
- DescartaTarjeta
 - fsm_rfid.c, 37
 - fsm_rfid.h, 55
- dest_state
 - fsm_trans_, 10
- err
 - player.c, 77
- EsperaTarjeta
 - fsm_rfid.c, 39
 - fsm_rfid.h, 55
- event
 - ISR_Typ, 11
 - InterruptSM.h, 64
- FLAG_BFF1_END
 - tipos.h, 82
- FLAG_BFF2_END
 - tipos.h, 82
- FLAG_END
 - tipos.h, 83
- FLAG_IRQ_STEPPER_CONTINUE
 - tipos.h, 83
- FLAG_IRQ_STEPPER_DIR
 - tipos.h, 83
- FLAG_IRQ_STEPPER_SELECT
 - tipos.h, 83
- FLAG_NOTA_TIMEOUT
 - tipos.h, 83
- FLAG_PLAYER_END
 - tipos.h, 84
- FLAG_PLAYER_START
 - tipos.h, 84
- FLAG_PLAYER_STOP
 - tipos.h, 84
- FLAG_QUIT
 - tipos.h, 84
- FLAG_START
 - tipos.h, 84
- FRAMES_PER_BUFFER
 - player.h, 79
- Final_Melodia
 - player.c, 73
- FinalizaReproduccion
 - fsm_rfid.c, 39
 - fsm_rfid.h, 56
- flag_fsm
 - tipos.h, 85
- flag_rfid
 - tipos.h, 85
- flags
 - BUFFERS, 6
- flags_player
 - tipos.h, 85
- flags_rfid
 - fsm_rfid.h, 51
- fp
 - player.c, 77
- fsm
 - fsm_audio_controller, 9
- fsm.c, 26
 - fsm_delete, 26
 - fsm_fire, 27
 - fsm_new, 27
- fsm.h, 27
 - fsm_delete, 29
 - fsm_fire, 29
 - fsm_input_func_t, 28
 - fsm_new, 29
 - fsm_output_func_t, 28
 - fsm_t, 28
 - fsm_trans_t, 28
- fsm_, 7
 - current_state, 8
 - tt, 8
 - user_data, 8
- fsm_audio_controller, 8
 - buffer, 9
 - fsm, 9
 - tipo_sistema, 9
- fsm_audio_controller_t
 - player.h, 80
- fsm_delete
 - fsm.c, 26
 - fsm.h, 29
- fsm_fire
 - fsm.c, 27
 - fsm.h, 29

- fsm_input_func_t
 - fsm.h, [28](#)
- fsm_new
 - fsm.c, [27](#)
 - fsm.h, [29](#)
- fsm_output_func_t
 - fsm.h, [28](#)
- fsm_rfid
 - fsm_rfid.h, [59](#)
- fsm_rfid.c, [30](#)
 - BuscaTarjeta, [32](#)
 - CancelaReproduccion, [33](#)
 - clean_list_files, [33](#)
 - ComienzaReproduccion, [34](#)
 - ComienzaSistema, [34](#)
 - CompruebaComienzo, [35](#)
 - CompruebaFinalReproduccion, [35](#)
 - CompruebaTarjeta, [36](#)
 - ConfiguraTarjeta, [36](#)
 - ConfiguracionCorrecta, [36](#)
 - DB_NAME, [32](#)
 - DescartaTarjeta, [37](#)
 - EsperaTarjeta, [39](#)
 - FinalizaReproduccion, [39](#)
 - get_last_file, [40](#)
 - get_list_files, [40](#)
 - get_next_file, [41](#)
 - get_number_files, [41](#)
 - ISR, [42](#)
 - killRFID, [43](#)
 - launchRFID, [43](#)
 - LeerTarjeta, [43](#)
 - lp, [44](#)
 - maquina_creada, [48](#)
 - menu_display_stepper_plus, [44](#)
 - new_list_files, [45](#)
 - PIN_A, [32](#)
 - PIN_B, [32](#)
 - PIN_C, [32](#)
 - select_mode, [45](#)
 - seqA, [48](#)
 - seqB, [49](#)
 - TarjetaDisponible, [46](#)
 - TarjetaNoDisponible, [46](#)
 - TarjetaNoValida, [47](#)
 - TarjetaValida, [47](#)
 - transition_table_rfid, [49](#)
 - UUID_2_int, [48](#)
 - UUID, [49](#)
- fsm_rfid.h, [50](#)
 - CancelaReproduccion, [52](#)
 - ComienzaReproduccion, [53](#)
 - ComienzaSistema, [53](#)
 - CompruebaComienzo, [54](#)
 - CompruebaFinalReproduccion, [54](#)
 - CompruebaTarjeta, [54](#)
 - DescartaTarjeta, [55](#)
 - EsperaTarjeta, [55](#)
 - FinalizaReproduccion, [56](#)
 - flags_rfid, [51](#)
 - fsm_rfid, [59](#)
 - launchRFID, [56](#)
 - LeerTarjeta, [56](#)
 - list_files_t, [51](#)
 - rfid_states, [52](#)
 - TarjetaDisponible, [57](#)
 - TarjetaNoDisponible, [57](#)
 - TarjetaNoValida, [58](#)
 - TarjetaValida, [58](#)
 - thread, [59](#)
 - UUID, [59](#)
- fsm_states
 - player.h, [80](#)
- fsm_t
 - fsm.h, [28](#)
- fsm_trans_, [9](#)
 - dest_state, [10](#)
 - in, [10](#)
 - orig_state, [10](#)
 - out, [10](#)
- fsm_trans_t
 - fsm.h, [28](#)
- func
 - player.c, [73](#)
- get_last_file
 - fsm_rfid.c, [40](#)
- get_list_files
 - fsm_rfid.c, [40](#)
- get_next_file
 - fsm_rfid.c, [41](#)
- get_number_files
 - fsm_rfid.c, [41](#)
- ISR_Typ, [11](#)
 - callback, [11](#)
 - event, [11](#)
 - pin, [11](#)
- ISR_Typ_InterruptSM.h, [63](#)
- ISR
 - fsm_rfid.c, [42](#)
- in
 - fsm_trans_, [10](#)
- Iniciliza_player
 - player.c, [74](#)
- InterruptSM.c, [60](#)
 - attachIsr, [61](#)
 - deleteIsr, [61](#)
 - loop, [62](#)
 - THRESHOLD_HIGH, [60](#)
 - THRESHOLD_LOW, [60](#)
- InterruptSM.h, [63](#)
 - atachPin, [65](#)
 - attachIsr, [64](#)
 - call_back, [63](#)
 - deleteIsr, [65](#)

- event, 64
 - ISR_Typ_, 63
 - threads, 65
- k
 - piMusicBox_2.c, 70
- killRFID
 - fsm_rfid.c, 43
- launchPlayer
 - player.c, 75
 - player.h, 81
- launchRFID
 - fsm_rfid.c, 43
 - fsm_rfid.h, 56
- LeerTarjeta
 - fsm_rfid.c, 43
 - fsm_rfid.h, 56
- lengthBuffer
 - BUFFERS, 7
- list_files, 12
 - current_file, 12
 - name_file, 12
 - num_files, 12
 - select_file, 13
- list_files_t
 - fsm_rfid.h, 51
- lock
 - mutex.c, 66
 - mutex.h, 68
- loop
 - InterruptSM.c, 62
- lp
 - fsm_rfid.c, 44
- mad_frame
 - player.c, 77
- mad_stream
 - player.c, 77
- mad_synth
 - player.c, 77
- main
 - piMusicBox_2.c, 70
- maquina_creada
 - fsm_rfid.c, 48
- menu_display_stepper_plus
 - fsm_rfid.c, 44
- mutex.c, 66
 - lock, 66
 - unlock, 66
- mutex.h, 68
 - lock, 68
 - unlock, 68
- name_file
 - list_files, 12
- new_buffer
 - player.c, 75
- new_list_files
 - fsm_rfid.c, 45
- num_file
 - tipos.h, 85
- num_files
 - list_files, 12
- orig_state
 - fsm_trans_, 10
- out
 - fsm_trans_, 10
- output
 - player.c, 76
- outputParameters
 - player.c, 78
- PIN_PWM
 - defines.h, 26
- PIN_A
 - fsm_rfid.c, 32
- PIN_B
 - fsm_rfid.c, 32
- PIN_C
 - fsm_rfid.c, 32
- PIN
 - tone.c, 88
- piMusicBox_2.c, 69
 - callback, 69
 - k, 70
 - main, 70
- piMusicBox_2.h, 71
- pin
 - ISR_Typ, 11
- player.c, 71
 - carga_bff1, 72
 - carga_bff2, 72
 - err, 77
 - Final_Melodia, 73
 - fp, 77
 - func, 73
 - Iniciliza_player, 74
 - launchPlayer, 75
 - mad_frame, 77
 - mad_stream, 77
 - mad_synth, 77
 - new_buffer, 75
 - output, 76
 - outputParameters, 78
 - stream, 78
 - transition_table_player, 78
- player.h, 79
 - BUFFERS_T, 80
 - FRAMES_PER_BUFFER, 79
 - fsm_audio_controller_t, 80
 - fsm_states, 80
 - launchPlayer, 81
 - SAMPLE_RATE, 80
- rfid_states
 - fsm_rfid.h, 52

SAMPLE_RATE
 player.h, 80
 SQL_QUERY_CHECK_DB
 dbcontroller.c, 16
 SQL_QUERY_CREATE_DB
 dbcontroller.c, 16
 SQL_QUERY_GET_NUMBER_OF_ELEMENTS
 dbcontroller.c, 16
 SQL_QUERY_INSERT_DB
 dbcontroller.c, 16
 SQL_QUERY_MAX_SIZE
 dbcontroller.c, 16
 SQL_QUERY_SELECT_DB
 dbcontroller.c, 16
 SQL_RESOLUTION
 dbcontroller.h, 22
 SQL_TABLE_NAME
 dbcontroller.c, 17
 sampleReaded
 BUFFERS, 7
 select_file
 list_files, 13
 select_mode
 fsm_rfid.c, 45
 seqA
 fsm_rfid.c, 48
 seqB
 fsm_rfid.c, 49
 song_name
 tipos.h, 85
 stepper_irq_flag
 tipos.h, 85
 stream
 player.c, 78

 THRESHOLD_HIGH
 InterruptSM.c, 60
 THRESHOLD_LOW
 InterruptSM.c, 60
 TarjetaDisponible
 fsm_rfid.c, 46
 fsm_rfid.h, 57
 TarjetaNoDisponible
 fsm_rfid.c, 46
 fsm_rfid.h, 57
 TarjetaNoValida
 fsm_rfid.c, 47
 fsm_rfid.h, 58
 TarjetaValida
 fsm_rfid.c, 47
 fsm_rfid.h, 58
 thread
 fsm_rfid.h, 59
 threads
 InterruptSM.h, 65
 tipo_sistema
 fsm_audio_controller, 9
 TipoSistema, 13
 tipos.h, 81

 FLAG_BFF1_END, 82
 FLAG_BFF2_END, 82
 FLAG_END, 83
 FLAG_IRQ_STEPPER_CONTINUE, 83
 FLAG_IRQ_STEPPER_DIR, 83
 FLAG_IRQ_STEPPER_SELECT, 83
 FLAG_NOTA_TIMEOUT, 83
 FLAG_PLAYER_END, 84
 FLAG_PLAYER_START, 84
 FLAG_PLAYER_STOP, 84
 FLAG_QUIT, 84
 FLAG_START, 84
 flag_fsm, 85
 flag_rfid, 85
 flags_player, 85
 num_file, 85
 song_name, 85
 stepper_irq_flag, 85
 toggle
 tone.c, 86
 tone.c, 86
 PIN, 88
 toggle, 86
 tone_init, 86
 tone_stop, 87
 tone_write, 87
 tone.h, 88
 tone_init, 88
 tone_stop, 89
 tone_write, 89
 tone_init
 tone.c, 86
 tone.h, 88
 tone_stop
 tone.c, 87
 tone.h, 89
 tone_write
 tone.c, 87
 tone.h, 89
 transition_table_player
 player.c, 78
 transition_table_rfid
 fsm_rfid.c, 49
 tt
 fsm_, 8

 UUID_2_int
 fsm_rfid.c, 48
 UUID
 fsm_rfid.c, 49
 fsm_rfid.h, 59
 unlock
 mutex.c, 66
 mutex.h, 68
 use_bcm
 defines.h, 26
 user_data
 fsm_, 8