Player Audio a distanza

Devin Taietta

16/06/2015

Sommario

Progetto Player Audio

This Software project is developed by Devin Taietta Last Edit mer 02 lug 2014 23:01:02 CEST This "Console" Project contains some open source library for all info see legal notes.

Introduzione

Questo progetto riguarda la realizzazione di un player Audio. L'obbiettivo iniziale è stato quello di realizzare i semplici comandi di pausa, successivo, precedente e lo studio base di come funziona l'audio contenuto nelle tracce(DSP). Una volta realizzato, si è deciso di sperimentare l'uso delle Socket studiate durante il quinto anno. E' stato quindi creato il sito client di interfacciamento e il **Server** per la gestione dei dati in arrivo. L'ultima implementazione è stata quella della trasmissione delle canzoni attraverso dispositivi Mobili come Smartphone e/o tablet attraverso lo strumento di sviluppo "Cordova" che permette di realizzare le applicazioni mobili per tutte le piattaforme.

Installazione

Per eseguire il file una volta eseguita la compilazione e' necessario avere installate le seguenti librerie:

LibAo -> Necessaria per scrivere sull'interfaccia audio

LibMpg123 -> Decoding (Ottenimento dai frammenti audio interlacciati e compressi di frammenti non compressi in formato PCM) dei file mp3

ffMpeg (avcodec,avformat) -> Decoding file audio in altri formati

Per la compilazione del progetto si usano gli strumenti gcc e g++. Se si vuole effettuare il debug dell'app allora si deve aggiungere l'opzione -g. E' inoltre necessaria l'opzione -std=c++0x che indica al compilatore che il progetto è stato realizzato con la nuova versione del C++ introdotta nel 2011 (Necessario per la libreria pthread del multithreading) Esempio di compilazione

```
 {.sh}
 gcc -c -g "Process/mpg123.c" "Output/aosender.c"
 g++ -g "aosender.o" "Input/usbData.cpp" "Input/Socket/Server.cpp"
"Process/dataManagerHub.cpp" "Process/util.cpp" "mpg123.o" "Process/Riproduttore.cpp"
"main_cli.cpp" -lao -lmpg123  -lavcodec -lavformat -std=c++0x -o AudioPlayer
```

Il sistema di versioning è gestito tramite Git. Si può vedere la lista dei commit digitando git log sul branch di sviluppo.

Successivamente è possibile eseguirlo digitando il nome dell'applicazione e il media esterno da tenere d'occhio.

```
 {.sh} AudioPlayer "/home/utente/musica/"
```

Hardware e Infrastrutture

Raspberry è una piattaforma embeded con processore ARM, da un costo contenuto con gli schemi OpenSource. Viene impiegata per l'uso Le comunicazioni invece avvengono tramite connessione WiFi (IEEE 802.11/g)

Struttura e codice del Server

Il **Server** viene sviluppato con il codice C (strutturato, linguaggio di medio livello) e C++ orientato agli oggetti. E' stato preferito questo linguaggio per il largo numero di librerie e per la sua velocità di esecuzione.

Le funzioni del server possono essere trovate sulla classe **Server**

Si veda anche:

Server

La gestione dell'archivio su supporto di memorizzazione esterna viene affidata alla classe **usbData**

Si veda anche:

usbData

La gestione invece dell'audio è affidata alla classe **Riproduttore**

Si veda anche:

Riproduttore

Il cordinamento di tutte le attività avviene tramite il dataManagerHub

Si veda anche:

dataManager

L'uso di tutte le classi e gli oggetti all'interno del programma è stato suddiviso secondo dove vanno ad operare. Abbiamo gli oggetti che si occupano di gestire l'input, quello che gestiscono il processamento dei dati e quelli che si occupano dei dati.

Struttura e codice della parte Web

La parte web è realizzata usando

HTML5: lingaggio di formattazione attraverso marcatori

Jquery(con Ajax) : Linguaggio derivato dal JavaScript che usa selettori sugli elementi DOM per effettuare le operazioni. AJAX è l'acronimo di Asynchronus Javascript and XML e viene usato per interagire con il server senza richiedere al client di ricaricare la pagina.

PHP: E' un linguaggio lato server viene usato nel progetto per interrogare il database e per gestire socket **Server** <--> Client

CSS (Bootstrap): Il framework per il CSS che viene usato è Bootstrap. il CSS rappresenta delle regole di formattazione di documenti HTML.

Struttura dell'app mobile

L'applicazione mobile è scritta in Java per Android o Objective C per IOS. Vengono sfruttate le webview per mostrare un sito con cui interfacciarsi al server.

"Come si usa?"

Da console

All'avvio dell'applicazione viene mostrato su schermo un menu che rappresenta tutte le operazioni svolgibili. Basta inserire il numero dell'operazione da svolgere per ottenere l'effetto desiderato.

Da sito web

Bisogna conoscere l'indirizzo IP del server per poter ottenere l'acccesso da web. Un impletazione futura vede uno schermo LCD dove sarà possibile attraverso pulsanti conoscere l'IP del server e configurare la connessione WiFi in maniera più personalizzata.ù Una volta acceduto attraverso un normale browser all'indirizzo sarà possibile interagire con l'applicazione attraverso una socket. ù

Da applicazione mobile

Per semplificare l'accesso al server viene fornita un applicazione client. Automaticamente è in grado di effettuare la scansione sugli host della rete una volta conosciuto il dominio. L'applicazione inoltre fornisce la possibilita di fare l'upload delle proprie canzoni in maniera da riprodurla sul **Server**.

Applicazioni pratiche

L'idea principale di impiego per cui si presta questo progetto è da applicarsi ad un locale dove si vede l'installazione di una cassa installata fissa da una parte (lontano da malintenzionati) con un

hardDisk integrato. A chi viene fornita la password di accesso (solitamente il personale, ma può essere estesa questa possibilità anche ai clienti) può comodamente da remoto scegliere la canzone da riprodurre o la playlist.

Altre applicazioni a cui si presta bene questo progetto o può essere facilmente adattato sono:

Una radio per una macchina: Devono essere aggiunti uno schermo e una radio FM. Adattando la parte esterna allo slot della macchina può essere realizzata una radio.

Player Audio grafico locale: Il progetto è stato realizzato in maniera da tenere CLI e Core dell'app separati, volendo sviluppare con Qt un applicazione grafica è possibile farlo.

future

Il progetto e' in continua evoluzione e vede molte funzioni da aggiungere:

Integrazione di API

Risveglio automatico

Indice dei tipi composti

Elenco dei tipi composti

Queste sono le classi, le struct, le union e le interfacce con una loro breve descrizione:

Indice dei file

Elenco dei file

Questo è un elenco dei file documentati con una loro breve descrizione:

Documentazione delle classi

Riferimenti per la classe dataManager

This class manage all the inputs and select the correct jobs or output.

`#include <dataManagerHub.h>`

Membri pubblici

dataManager (std::string dir)

*Costruttore della classe **dataManager**:*

void **setMenu** (bool status)

void **setStream** (bool status)

Attiva o disattiva il Menu da terminale.

bool **killSignal** ()

~dataManager ()

Controlla se è arrivato il segnare di uscita.

Membri privati

void **menu_thr** ()

Thread del Menu.

void **data_thr** ()

*Thread che analizza ciò che arriva da **Server**,Raspberry e CLI ed esegue l'azione più opportuna.*

void **sendSongsToWeb** ()

Funzione che invia in formato JSON le canzoni presenti in Archivio.

Attributi privati

bool **menuActive**

Variabile booleana che mostra il menù del terminale o no. Và impostata tramite la funzione.

bool **streamActive**

Variabile booleana che memorizza se il servizio di streaming delle canzoni da dispositivi è attivo o no. Viene impostata a false nel costruttore. Per modificarla vedi setStream.

bool **dataManagerQuitSignal**

Variabile controllata nei loop sempre attivi fino alla fine del programma.

Descrizione dettagliata

This class manage all the inputs and select the correct jobs or output.

La parte restante del buffer è il nostro messaggio che verrà interpretato dal thread **data_thr()**. In generale:

1. Se viene dall'interfaccia WEB, da Raspberry o dalla CLI, verrà interpretato come comando per il **Riproduttore**
2. Se viene dall'applicazione per Android la canzone verrà caricata in un nuovo riproduttore. Alla disconnessione verrà ripristinato il puntatore al vecchio riproduttore

Autore:

Devin Taietta

Definizione alla linea 31 del file dataManagerHub.h.

Documentazione dei costruttori e dei distruttori

dataManager::dataManager (std::string *dir*)

Costruttore della classe **dataManager**:

Viene richiamato al momento della creazione di un istanza della classe. std::string

Parametri:

dir,:	E' un parametro obbligatorio dove deve essere indicato il percorso iniziale o della pennetta o di una cartella contenente la musica

dataManager::~dataManager ()

Controlla se è arrivato il segnare di uscita.

Distruttore della classe **dataManager**. In più pulisce la memoria occupata nell'heap e restituisce un messaggio di arrivederci.

Definizione alla linea 261 del file dataManagerHub.cpp.

Documentazione delle funzioni membro

void dataManager::data_thr () [private]

Thread che analizza ciò che arriva da **Server**,Raspberry e CLI ed esegue l'azione più opportuna.

Controllare che ci sia almeno un processo attivo!

Definizione alla linea 58 del file dataManagerHub.cpp.

void dataManager::setStream (bool *status*) [inline]

Attiva o disattiva il Menu da terminale.

Parametri:

Definizione alla linea 23 del file dataManagerHub.cpp.

status,:	TRUE=attivo FALSE=disattivo(default). Imposta la variabile che ci indica se il servizio di streaming da tablet è attivo o no.

Documentazione dei membri dato

bool dataManager::menuActive [private]

Variabile booleana che mostra il menù del terminale o no. Và impostata tramite la funzione.

Si veda anche:

setMenu()

Definizione alla linea 52 del file dataManagerHub.h.

bool dataManager::streamActive [private]

Variabile booleana che memorizza se il servizio di streaming delle canzoni da dispositivi è attivo o no. Viene impostata a false nel costruttore. Per modificarla vedi setStream.

Si veda anche:

setStream()

Definizione alla linea 53 del file dataManagerHub.h.

La documentazione per questa classe è stata generata a partire dai seguenti file:

1 Process/**dataManagerHub.h**
2 Process/**dataManagerHub.cpp**

Riferimenti per la classe Riproduttore

Classe **Riproduttore** usata per gesitre le azioni sulle canzoni, scegliendo la libreria più adatta.
`#include <Riproduttore.h>`

Membri pubblici

Riproduttore (std::string percorso, bool recursive, bool autoskip)
void **update** (std::string percorso="")
void **scegliCanzone** (int posizione)
void **prevSong** ()
*La funzione **prevSong()** viene usata per scegliere la canzone precedente e riprodurla.*
void **nextSong** ()
*La funzione next **nextSong()** viene usata per scegliere la canzone successiva e riprodurla.*
void **playPause** (int status=9)
void **stop** ()
Stoppa la riproduzione dell'audio.
void **setTrackEnded** (bool value)
Imposta l'attuale stato della traccia.
bool **getTrackEnded** ()
Ottiene se l'attuale traccia è terminata o no.
void **lock** ()
Set the thread in wait.
void **unlock** ()
Set the thread in resume.
void **kill** ()
Kill the player(so the thread)
~Riproduttore ()
Riproduttore (const **Riproduttore** &other)
Riproduttore & **operator=** (const **Riproduttore** &other)
Overload dell'operatore di copia, necessario perchè senno il mutex non è copiabile.

Attributi pubblici

usbData * **Musica**
*Puntatore "Musica" per un oggetto di tipo **usbData**.*
std::mutex **lockaudio**

Membri privati

void **riproduttore_thr** ()

Attributi privati

bool **loopSongs**
bool **endtrack**
*Indica se la traccia è attualmente finita. */.*
bool **locked**
*Variabile usata per mettere in pausa il processo. */.*

bool **RiproduttoreQuitSignal**

*Se impostata a Vero termina si termina il processo riproduttore_thr e si chiama il distruttore. */.*

Descrizione dettagliata

Classe **Riproduttore** usata per gesitre le azioni sulle canzoni, scegliendo la libreria più adatta. Libreria per il multithetering Blocco sezioni critiche !check or implement in the right way Definizione alla linea 17 del file Riproduttore.h.

Documentazione dei costruttori e dei distruttori

Riproduttore::Riproduttore (std::string *percorso*, bool *recursive*, bool *autoskip*)

La funzione Riproduttore::Riproduttore corrisponde al costruttore dell'ononima classe. Prende in ingresso due parametri e si occupa di: 1) Inizializzare le variabili private 2) Analizzare un percorso di dati per trovare tutti i file audio al suo interno 3) Impostare il puntatore a 0 (Prima traccia) 4) Far partire in background il processo che gestisce l'audio

Stringa

Parametri:

Definizione alla linea 41 del file dataManagerHub.h.

percorso	Cartella da analizzare per scoprire le tracce audio. Booleano
recursive	Impostare a vero se si vuole ottenere la scansione delle **sotto-directory** .

Non restutuisce nulla.

Definizione alla linea 20 del file Riproduttore.cpp.

Riproduttore::~Riproduttore ()

Il distruttore si occuperà di: 1)liberare la memoria occupata dal vettore contenente le posizioni 2)liberare la memoria occupata dal decoder 3)Terminare il processo di riproduzione dei frammenti audio 4)Eliminare la classe

Definizione alla linea 122 del file Riproduttore.cpp.

Documentazione delle funzioni membro

void Riproduttore::playPause (int *status* = 9)

La funzione playPause serve per gestire lo stato del lettore. Si può quindi decidere se stiamo in riproduzione o in pausa

Intero

Parametri:

Restituisce:

in	*status.*	Può assumere questi valori: **9** (Predefinito): Dopo aver controllato che la traccia non è finita, riproduci il frammento Audio **0** : Metti in

		pausa il decoder audio **1** : Metti in Play il decoder audio **Altro:** Se viene impostato un altro numero (Es. 3) il decoder inverte il suo stato tramite il flip/flop della variabile booleana

void Riproduttore::prevSong ()

La funzione **prevSong()** viene usata per scegliere la canzone precedente e riprodurla.

Riproduttore::playPause();
Definizione alla linea 73 del file Riproduttore.cpp.

void Riproduttore::setTrackEnded (bool *value*)

Imposta l'attuale stato della traccia.

Parametri:
Definizione alla linea 91 del file Riproduttore.cpp.

value	(Booleano) Impostare a TRUE per indicare la fine della traccia

void Riproduttore::update (std::string *percorso* = "")

La funzione update ha duplice scopo: con

Parametri:
Definizione alla linea 49 del file Riproduttore.cpp.

percorso	(Stringa) impostato si occupa di riprodurre esattamente il file che gli viene passato. Senza parametri inseriti, quindi con percorso valore "", la funzione verifica se ciò che arriva è un MP3 o un altro file audio e chiama il corretto Reset

resetMp3()
resetAudio()

Parametri:

Si veda anche:

recursive	(booleano): Indica se si deve effettuare la ricerca anche nelle sottocartelle

---

Documentazione dei membri dato

std::mutex Riproduttore::lockaudio `[mutable]`

Usato nelle sezioni critiche per ottenere l'accesso esclusivo al processo
Definizione alla linea 62 del file Riproduttore.h.

usbData* Riproduttore::Musica

Puntatore "Musica" per un oggetto di tipo **usbData**.

Si veda anche:

usbData.*/

Definizione alla linea 32 del file Riproduttore.h.

La documentazione per questa classe è stata generata a partire dai seguenti file:

3 Process/**Riproduttore.h**

4 Process/**Riproduttore.cpp**

Riferimenti per la classe Server

Questa classe gestisce la SocketServer e le socket di comunicazione tra dispositivi e questo programma.

`#include <Server.h>`

Membri pubblici

Server (int porta)
void **SetactiveThread** (bool status)
bool **getActived** ()
*Indica se il **Server** è attivo.*
bool **isConn** ()
Test della connessione al client.
void **sendData** (std::string **message**, std::string type)
void **getData** ()
int **getApp** ()
std::string **getMessage** ()
~Server ()
*Distruttore della classe **Server**.*

Membri privati

void **ServerThread** ()
Processo che gira in continuazione in attesa di nuove connessioni.

Attributi privati

bool **conn**
Variabile privata che indica se attualmente il client è connesso.
bool **kill**
Variabile privata che indica se il processo deve essere killato.
bool **thrActive**
int **currentApp**
Mantiene in memoria fino a che non viene richiesta l'app corrente.
int **startStop**
int **msgLen**
std::string **message**
Mantiene il messaggio fino a che non è richiesto.

Descrizione dettagliata

Questa classe gestisce la SocketServer e le socket di comunicazione tra dispositivi e questo programma.

Definizione alla linea 8 del file Server.h.

Documentazione dei costruttori e dei distruttori

Server::Server (int *porta*)

Costruttore della classe Server, si occupa quindi di inizializzare il server, lanciare il thread di accettazione della connessione e impostare le variabili.

Parametri:

Definizione alla linea 51 del file Riproduttore.cpp.

porta	dove far partire il server. Valori accettabili sono **1024-49151** . Sotto è sconsigliato l'uso di qualche porta e può richiedere diritti di amministrazione.

Documentazione delle funzioni membro

bool Server::getActived ()

Indica se il **Server** è attivo.

Restituisce:

bool value.
Se True il thread del server è attivo e viceversa

Definizione alla linea 57 del file Server.cpp.

int Server::getApp ()

Restituisce:

currentApp: Valore intero che rappresenta il tipo di applicazione client o locale che ha mandato il Messaggio

Si veda anche:

data_thr()

Definizione alla linea 55 del file Server.cpp.

void Server::getData ()

Fà una prima analisi riempendo le oppurtune variabili private.

Funzionamento del Codice:

Codice[0]: Il primo numero contiene la /b priorità; E' un numero da 0 a 2 che indica se il comando attuale deve passare sopra ad un altro.

Questo codice verrà inserito nella variabile "Priority"

Si veda anche:

getPriority()
Codice[1]: Indica l'applicazione che ha mandato il codice

0: CLI
1: Tasti su Raspberry
2: Sito Web
3: Applicazione Web Questo codice verrà inserito nella variabile "App"

getApp()
Codice[2]: Indica l'inizio e la fine di un set di comandi. Ad esempio se stiamo riproducendo un buffer rappresenta l'inizio e la fine dello streaming Il messaggio viene recuperato tramite la funzione **ottieniData()** in **ServerUtil.cpp**

dataManager::data_thr()
Analisi del codice

Definizione alla linea 32 del file Server.cpp.

std::string Server::getMessage ()

Restituisce:

message: Il testo che si ottiene togliendo le altre parti del codice. Questo verrà successivamente analizzato e usato nello scopo più adatto

Definizione alla linea 60 del file Server.cpp.

bool Server::isConn ()

Test della connessione al client.

Restituisce:

conn: Valore booleano che indica se si è connessi

Definizione alla linea 56 del file Server.cpp.

La documentazione per questa classe è stata generata a partire dai seguenti file:

5 Input/Socket/**Server.h**
6 Input/Socket/**Server.cpp**

Riferimenti per la classe usbData

usbData Class: Analizza cartelle in cerca di file e li carica in un vettore. Mantiene inoltre il puntatore che ci indica a quale canzone ci troviamo

`#include <usbData.h>`

Membri pubblici

usbData (std::string baseDir, bool recursive)

Costruttore. Analizza la cartella e setta il puntatore.

.

void **setPuntatore** (int posizione)

Imposta la posizione al puntatore

.

int **getPuntatore** ()

Ottieni la posizione del puntatore.

bool **isAudio** (std::string percorso, bool onlymp3)

Controlla l'estensione per capire se ci si trova di fronte ad un file audio.

usbData & **operator=** (const **usbData** &other)

Attributi pubblici

std::vector< std::string > **elementi**

Tutte le canzoni trovate

.

Membri privati

bool **isDir** (std::string dir)

Controlla se è una cartella.

void **riempiArchivio** (std::string baseDir, bool recursive, **usbData** &arc)

Funzione interna per riempire l'archivio(in realtà il vettore) delle canzoni.

Attributi privati

int **puntatore**

Descrizione dettagliata

usbData Class: Analizza cartelle in cerca di file e li carica in un vettore. Mantiene inoltre il puntatore che ci indica a quale canzone ci troviamo

Definizione alla linea 12 del file usbData.h.

Documentazione dei costruttori e dei distruttori

usbData::usbData (std::string *baseDir*, bool *recursive*)

Costruttore. Analizza la cartella e setta il puntatore.

.

Parametri:

Definizione alla linea 18 del file Server.cpp.

baseDir	Directory iniziale
recursive	Impostare a vero se si vogliono analizzare anche le sottocartelle

---

Documentazione delle funzioni membro

int usbData::getPuntatore ()

Ottieni la posizione del puntatore.

Restituisce:

puntatore Tipo Intero. Il Puntatore serve per sapere a quale canzone del vettore ci troviamo. **Non** è inteso come puntatore di programmazione ma nel senso comune di "Indicatore"

Definizione alla linea 35 del file usbData.cpp.

bool usbData::isAudio (std::string *percorso*, bool *onlymp3*)

Controlla l'estensione per capire se ci si trova di fronte ad un file audio.

Parametri:

Definizione alla linea 18 del file usbData.cpp.

percorso	
onlymp3	se impostato a true limita la funzione al solo controllo dell'estensione degli mp3.

bool usbData::isDir (std::string *dir*) `[private]`

Controlla se è una cartella.

Parametri:

Definizione alla linea 51 del file usbData.cpp.

dir	La cartella da analizzare

bool vero se è una cartella o viceversa

Definizione alla linea 72 del file usbData.cpp.

void usbData::setPuntatore (int *posizione*)

Imposta la posizione al puntatore

.

Parametri:

Restituisce:

posizione	

Documentazione dei membri dato

std::vector<std::string> usbData::elementi

Tutte le canzoni trovate

.

Definizione alla linea 23 del file usbData.h.

La documentazione per questa classe è stata generata a partire dai seguenti file:

7 Input/**usbData.h**
8 Input/**usbData.cpp**

Documentazione dei file

Riferimenti per il file Input/Socket/Server.cpp

```
#include "ServerUtil.cpp"
#include "Server.h"
#include <iostream>
#include <string>
#include <thread>
#include <mutex>
#include <condition_variable>
```

Descrizione dettagliata

Definizione nel file **Server.cpp**.

Riferimenti per il file Input/Socket/Server.h

```
#include <iostream>
```

Composti

class **Server**

> *Questa classe gestisce la SocketServer e le socket di comunicazione tra dispositivi e questo programma.*

Descrizione dettagliata

Server.h - Class **Server**

Definizione nel file **Server.h**.

Riferimenti per il file Input/Socket/ServerUtil.cpp

```
#include <iostream>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
```

Funzioni

void **mandaData** (char *data, int len)
char * **ottieniData** ()

Funzione Statica OttieniData()

void **acceptClient** ()
void **closeServer** ()
void **killServer** ()
std::string **JSONEncode** (std::string message, std::string type)

Descrizione dettagliata

(EN) A simple server in the internet domain using TCP.

(IT) Un semplice server nel dominio internet che usa il TCP (Protocollo orientato alla connessione)

Definizione nel file **ServerUtil.cpp**.

Documentazione delle funzioni

void mandaData (char * *data*, int *len*)

Restituisce al Client una risposta

Parametri:

Definizione alla linea 34 del file usbData.cpp.

data	Puntatore a variabile char che contiene i dati
len	Intero, lunghezza dei dati da trasmettere

char* ottieniData ()

Funzione Statica OttieniData()

Restituisce:

Puntatore a un indirizzo contenente Char. (Char*).

Definizione alla linea 69 del file ServerUtil.cpp.

Riferimenti per il file Input/usbData.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include <errno.h>
#include "usbData.h"
#include "/usr/include/taglib/fileref.h"
#include "/usr/include/taglib/tag.h"
#include <dirent.h>
#include <sys/stat.h>
```

Descrizione dettagliata

Definizione nel file **usbData.cpp**.

Riferimenti per il file Input/usbData.h

```
#include <iostream>
#include <vector>
#include <string>
```

Composti

class **usbData**

usbData *Class: Analizza cartelle in cerca di file e li carica in un vettore. Mantiene inoltre il puntatore che ci indica a quale canzone ci troviamo*

Descrizione dettagliata

Definizione nel file **usbData.h**.

Riferimenti per il file main_cli.cpp

```
#include <iostream>
#include <string>
#include "main_cli.h"
#include "Process/dataManagerHub.h"
```

Funzioni

int **main** (int argc, char *argv[])

Descrizione dettagliata

Definizione nel file **main_cli.cpp**.

Riferimenti per il file Output/aosender.c

```
#include <ao/ao.h>
#include "aosender.h"
```

Funzioni

void **initAudioDev** (int bits, long rate, int channels)
void **freeAudioDev** ()
int **writeAudio** (char *buffer, uint_32 numBytesMemory)

Descrizione dettagliata

Definizione nel file **aosender.c**.

Riferimenti per il file Process/dataManagerHub.cpp

```
#include "dataManagerHub.h"
#include "../Input/Socket/Server.h"
#include "../Process/Riproduttore.h"
#include "util.h"
#include <iostream>
#include <string>
#include <cstring>
#include <thread>
#include <mutex>
#include <condition_variable>
```

Variabili

std::mutex **lockAction**
int **action** = -1
int **app** = -1
const std::string **dirUpload** = "WebManager/uploadedFiles/"
Server comandiServer (7777)
Riproduttore * **Audio**
Riproduttore * **disabledPlayer**

Descrizione dettagliata

Definizione nel file **dataManagerHub.cpp**.

Riferimenti per il file Process/dataManagerHub.h

```
#include <iostream>
```

Composti

class **dataManager**

This class manage all the inputs and select the correct jobs or output.

Descrizione dettagliata

(EN) This file manage all the inputs and select the correct jobs or output (IT) Questo file gestisce tutti gli input e seleziona il giusto lavoro da eseguire.

Nella fase di **input** Analizza i dati che gli arrivano seguendo questo codice:

Nella fase di **risposta da server** a **client** , risponde usando la codifica JSON. anche la lista delle canzoni viene mandata in formato JSON per essere inserita poi nel database WEB.

Manda inoltre i comandi alla classe/funzione Streaming che gestirà la scrittura su scheda Audio

Manda comandi e informazioni al gestore delle informazioni che si occupa di scegliere il mezzo più appropriato per notificare quella particolare informazione all'utente. Teoria codici passati per la comunicazione ---------- Si è voluto cercare un sistema per permettere una comunicazione efficace tra il client e il server. Il problema principale è nato dal fatto che il comportamento non è standard ma dipende dall'applicazione che ha generato i dati che vengono inviati attraverso la Socket.

Il **primo** numero in arrivo corrisponde all'applicativo che ha mandato il segnale

1. **0** Rappresenta l'interfaccia mostrata a riga di comando
2. **1** Rappresenta i tasti comunicanti attraverso la GPIO (General Purpose Input/Output) di Raspberry
3. **2** Rappresenta l'interfaccia WEB presente sul server Apache installato sulla macchina dove risiede il programma
4. **3** Rappresenta l'applicazione JAVA installabile su dispositivi Android che invia direttamente il frame da riprodurre

Definizione nel file **dataManagerHub.h**.

Riferimenti per il file Process/ffmpeg.cpp

```
#include "ffmpeg.h"
#include <libavutil/mathematics.h>
#include <libavformat/avformat.h>
#include <libswscale/swscale.h>
#include <libavcodec/avcodec.h>
#include "../Output/aosender.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
```

Definizioni

#define **AUDIO_INBUF_SIZE** 20480

Funzioni

void **die** (const char *msg)
void **resetAudio** (char *song)
int **playAudio** ()
void **freeAudio** ()
void **setAudioStatus** (int value)

Variabili

int **FFMPEG_isPaused**
const int **buffer_size** = AUDIO_INBUF_SIZE + FF_INPUT_BUFFER_PADDING_SIZE

Descrizione dettagliata

Definizione nel file **ffmpeg.cpp**.

Riferimenti per il file Process/ffmpeg.h

Funzioni

void **resetAudio** (char *song)
int **playAudio** ()
void **freeAudio** ()
void **setAudioStatus** (int value)

Descrizione dettagliata

Devin Taietta - **ffmpeg.h** Riproduzione file Flac,Wav

Definizione nel file **ffmpeg.h**.

Riferimenti per il file Process/mpg123.c

```
#include <mpg123.h>
#include "../Output/aosender.h"
```

Funzioni

void **resetMp3** (char *song)

Inizializza il decoder

.

int **playMp3** ()

Riproduce un frammento d'Audio.

void **freeMp3** ()

Pulisce l'heap dai puntatori creati e resetta le variabili.

void **setMp3Status** (int value)

Imposta la variabile status. 0=Pausa, 1=Play /n.

Variabili

const int **BITS** = 8

int **isPaused** = 0

Descrizione dettagliata

Definizione nel file **mpg123.c**.

Documentazione delle funzioni

void resetMp3 (char * *song*)

Inizializza il decoder

.

Parametri:

Definizione alla linea 55 del file ServerUtil.cpp.

song	Percorso del file da leggere

void setMp3Status (int *value*)

Imposta la variabile status. 0=Pausa, 1=Play /n.

Parametri:

Definizione alla linea 18 del file mpg123.c.

value	Valore intero da impostare.

Riferimenti per il file Process/mpg123.h

Funzioni

void **resetMp3** (char *song)

Inizializza il decoder

.

int **playMp3** ()

Riproduce un frammento d'Audio.

void **freeMp3** ()

Pulisce l'heap dai puntatori creati e resetta le variabili.

void **setMp3Status** (int value)

Imposta la variabile status. 0=Pausa, 1=Play /n.

Descrizione dettagliata

mpg123.h Devin Taietta - Riproduzione file MP3 -

Definizione nel file **mpg123.h**.

Documentazione delle funzioni

void resetMp3 (char * *song*)

Inizializza il decoder

.

Parametri:

Definizione alla linea 72 del file mpg123.c.

song	Percorso del file da leggere

void setMp3Status (int *value*)

Imposta la variabile status. 0=Pausa, 1=Play /n.

Parametri:

Definizione alla linea 18 del file mpg123.c.

value	Valore intero da impostare.

Riferimenti per il file Process/Riproduttore.cpp

```
#include <vector>
#include "Riproduttore.h"
#include "../Input/usbData.h"
#include "util.h"
#include "ffmpeg.h"
#include "mpg123.h"
#include <thread>
#include <mutex>
#include <condition_variable>
```

Descrizione dettagliata

Definizione nel file **Riproduttore.cpp**.

Riferimenti per il file Process/Riproduttore.h

```
#include "../Input/usbData.h"
#include <iostream>
#include <string>
#include <thread>
#include <mutex>
```

Composti

class **Riproduttore**

*Classe **Riproduttore** usata per gesitre le azioni sulle canzoni, scegliendo la libreria più adatta.*

Descrizione dettagliata

(EN) This file allow us to do basic action on song like skip current track or pause current track.

(IT) Questo file ci consente di eseguire delle azioni sulle canzoni come passare alla precedente o successiva traccia o mettere la musica in pausa.

Autore:

Devin Taietta

Definizione nel file **Riproduttore.h**.

Riferimenti per il file WebManager/filesUploader.php

Variabili

$new_song_name = "uploadedSong.mp3"

Descrizione dettagliata

Definizione nel file **filesUploader.php**.