





Table of Contents

[Abstract](#h.gjdgxs)

[Introduction](#h.30j0zll)

[Implementation](#h.1fob9te)

[Overview](#h.3znysh7)

[Hardware](#h.2et92p0)

[Circuitry](#h.tyjcwt)

[Software](#h.3dy6vkm)

[Code](#h.1t3h5sf)

[Results](#h.4d34og8)

[Future Ideas](#h.2s8eyo1)

[Appendices](#h.17dp8vu)

[Appendix A](#h.3rdcrjn)

[Appendix B](#h.26in1rg)

[Appendix C](#h.lnxbz9)

[Appendix D](#h.35nkun2)

# Abstract

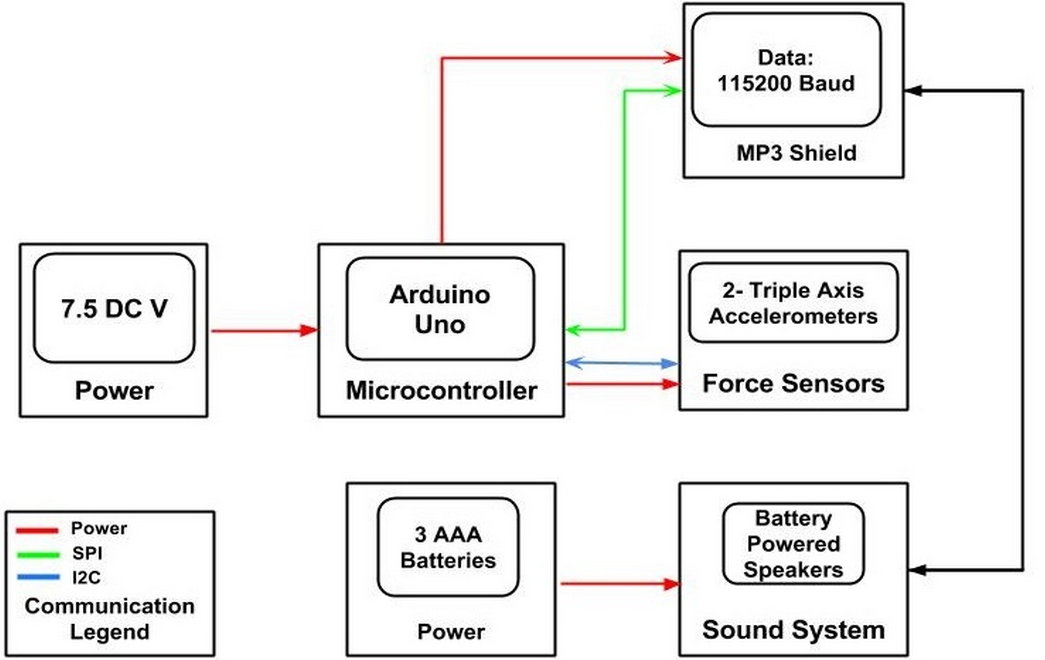
# Introduction

Music should not only be controlled by buttons on a screen, music is about motion and should be controlled by motion as well. The purpose of this project is to create a wearable music playing device that allows the user to adjust settings of music using their hands. This technology can be used by dancers who want to be able to change their music they are listening to without having to get their music player out of their pocket or pause their dancing. The Dynamic DJ will read input from the user’s hands and given a certain combination of motions, the settings of the music will be changed accordingly.

# Implementation

## Overview

Accelerometers were used on the hands of the user to detect motion. These accelerometers were hooked up to the Arduino. An Mp3 Shield was attached to the Arduino to play music loaded on an SD card. Any set of headphones or speakers with a headphone plug can be plugged into the Mp3 Shield to listen to the music (Figure 1). The controls of the Dynamic DJ are in Appendix B.



*Figure 1 General Functional Block Diagram*

## Hardware

### Circuitry

The key components of the circuit used for the Dynamic DJ were the MP3 Shield, accelerometers, and the Arduino RED board. As a prototype build, most of the Dynamic DJ circuitry is implemented on a bread board held in place by a black mounting plate that also hold the Arduino.

The MP3 Shield is powered through the Arduino and communicates directly using SPI protocol. The MP3 Shield has various connections with the Arduino (Appendix C) leaving a couple of digital pins (pins 5 and 10), all analog pins, I2C communication pins, and only a couple others open. The accelerometers are able to use the free SDA and SCL pins to communicate with the Arduino using I2C protocol. The accelerometers used were MMA8542 triple axis accelerometers that were purchased on break out boards. The break out boards were easy to connect and the only modifications needed were to change the address of one accelerometer in order to have both accelerometers communicate with the Arduino at the same time.

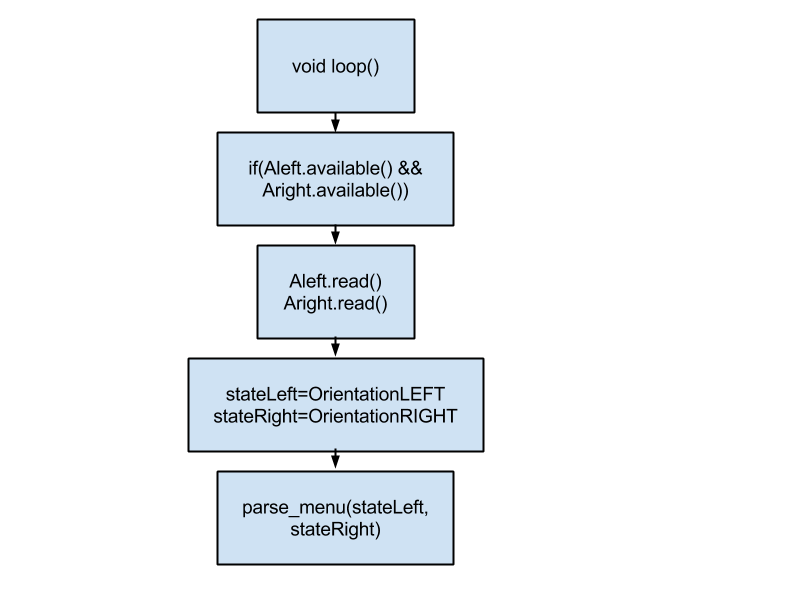
A full schematic of the Dynamic DJ circuit can be seen in Appendix C.

## Software

### Code

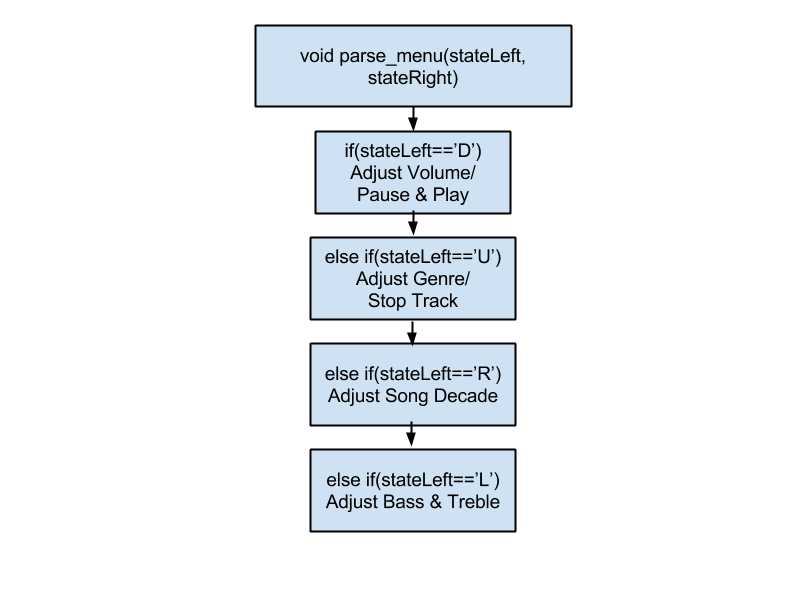
Code was written into the program to collect data from the accelerometers to detect the orientation of the hands. Given a certain combination of orientations on the left and right hand, certain settings of the music will be changed. Boolean operators were used to insure one setting wouldn’t get changed more than the user desires. Each time the right hand goes to the flat orientation, the Boolean operator is assigned the value of true. Once the hand changes orientation and the Boolean operator is true, the setting is changed and the Boolean operator is assigned false so the change in settings will not be repeated.

Code was included in the program to change the settings of the music based on the position of the user’s hands. This code interacted with the MP3 Shield and the SD card containing the music. The code began as Bill Porter’s library for playing music from an SD card on the MP3 Shield. This code was used to adjust volume, adjust bass, adjust treble, play, pause, and stop music. Using the ID3 tags to find the genre and year of the song, the songs on the SD card will be sorted so the user can select a certain genre or decade of song they would like to listen to. Code to read the genre and year was written into the MP3 library in order to make coding more efficient.



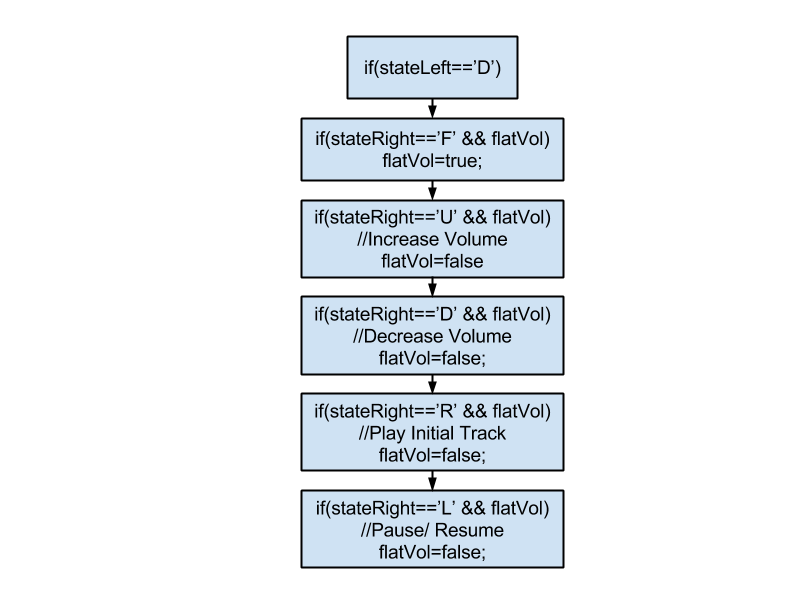
*Figure 2 Flow chart of main loop*

In the loop of the program, the program is constantly polling for information from the accelerometers. If information is available, it reads the orientations of the accelerometers and stores it in the variables *stateLeft* and *stateRight*. Parse menu is called using these variables and then the loop repeats. (Figure 2).



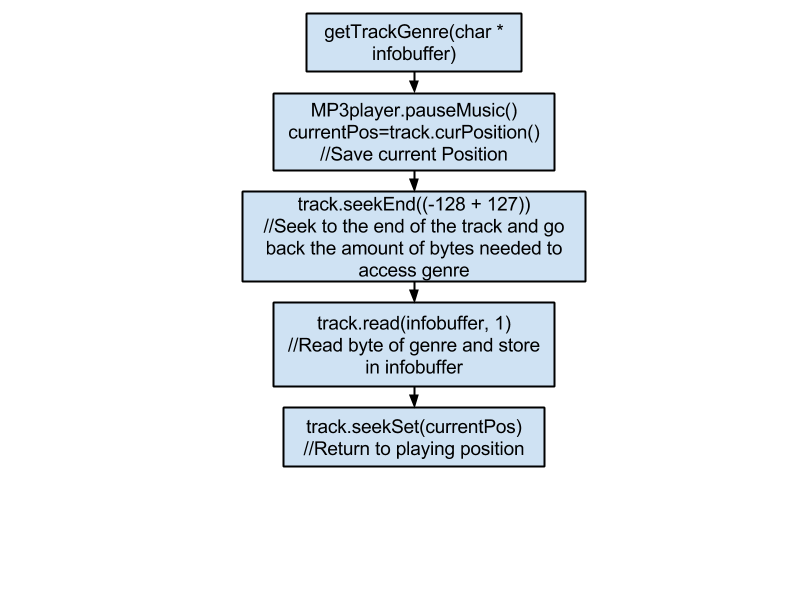
*Figure 3 Flow chart of the parse\_menu function*

The parse menu is a series of if and else if statements based on the orientation of the left hand which changes the mode as shown above (Figure 3).



*Figure 4 Flow chart of left hand controls*

In each mode outlined in the Parse Menu a structure similar to the above layout is present. When the right hand is flat, the Boolean operator is set to true. When the orientation of the hand changes, the function to change the settings of the music is executed and then the Boolean operator is set to false so the setting will not keep changing when the user does not intend the MP3 to do so (Figure 4).



*Figure 5 Flow chart of getTrackGenre function*

The function to read the genre was implemented by first pausing the music and saving the position at that point. The ID3 tag information is stored at the very end of the MP3 file. The last 128 bytes of the file is this information. The genre is the very last byte of this information so the function seeks back to that byte and reads one byte, storing the information in address pointed to by *infobuffer*. The track then seeks back to the position initially saved and resumes playing (Figure 5).

The same method was used to read the year except the function will seek back to -128+93, since the year is stored 93 bytes from the end of the file. The read function will read four bytes instead of four since the year is stored in four bytes.

# Testing

#### Hardware

To test the accelerometers, headers were soldered to the break out boards so that the accelerometers could be tested on a bread board. The first step was to connect one accelerometer to the Arduino (without the MP3 shield) and to try and get correct outputs in order to understand the range of states it could detect. The next step was to add the second accelerometer and try to get both accelerometers reading different states and responding differently for each combination. To do this, a simple function was made to output orientations to the serial monitor. Once the accelerometers were working, arm length wires were soldered to each accelerometer for testing with movement.

The testing for the MP3 shield was mostly done from previous homework assignments in class. The shield was testing again, however, to make sure that the MP3 shield could output properly to speakers instead of ear buds.

The most important step was to combine the music with the movement. First, the MP3 shield was added to the accelerometer circuit and basic accelerometer functions were added to the music playing code to see if the two modules could run together. Once this was confirmed, testing continued by altering both types of code until the Dynamic DJ ran as planned.

#### Software

Software and hardware components were developed separately until the final stage of the project. Testing of the software involved hooking up an MP3 Shield with an SD card to an Arduino and connecting the Arduino to a computer. Code was added to the program to print the results of the year and genre to ensure results of the genre and year. The other commands such as volume, bass, treble, pause, and play were tested using commands from the serial monitor instead of input from accelerometers.

Once the software and hardware were combined, a speaker was plugged into the MP3 Shield and a tester used the hand motions to test if the function was working. The code was then changed according to the problems that occurred and testing continued.

# Technical Difficulties

#### Hardware

Testing the Dynamic DJ with arm length wires was difficult because the wires were too stiff. The body movements kept pulling out the wires from the bread board and breaking the solder connections on the accelerometers. To fix this, the wires near the bread board were held down the by straps the Dynamic DJ. This minimized the strain on the connections by the movement of the user.

#### Software

The MP3 files loaded onto the SD card had ID3v2 tags. Research did not supply sufficient information about the structure of the ID3v2 information. This was a necessary part of the project. A program was downloaded that converted the metadata tags of the file from ID3v2 to ID3v1. The structure of the ID3v1 tag is much more accessible and the information was read by accessing the bytes of information in the MP3 file.

An original goal was to be able to skip to the next track. There were difficulties finding where the file name of the track was stored so that the track name could be incremented and the next track would be played. This problem did not get resolved within this project, instead a function was added to stop the track and then the user could play another. The problem of skipping tracks could be addressed at a future time.

# Results

The end product was much more basic and direct than our initial goals. We originally intended to create a music-filling device according to ambiance mainly by detecting movement amongst people. Our ultimate goal was to detect dance moves from users, chart data comparing our perceived intention of mood and play songs with suitable genres or tempo fitting the perceived.

Through much discussion with our professor and teaching assistant we realized that using solely accelerometers would not fulfill our initial intentions. Having discussed rate gyros and other parts we decided it was not within our budget or time frame to add more devices.

The newly designed functions of the Dynamic DJ are based on the capacities within an accelerometer and the extent of our knowledge on metadata ID3 tags within MP3 files. Creating, modifying, and editing software to play MP3 files was an exceedingly greater task than executing our intended hand motions into input for our program. However in retrospect, we did considerably simplify our range of hand movements to create a more attainable goal within the duration of this project.

Creating user-friendly functions to adjust music settings with smooth transitions from the accelerometers’ sensors into registers that adjust the music was eminently our greatest task. The several combinations of adjustable music settings are very sensitive to hand movement and must be direct. Without a visual screen, it is extremely presumptuous to assume any user will accurately memorize all the hand gestures in accordance with their setting, specific hand movement, and step-by step method needed to carry-out a task.

In its current state, the Dynamic DJ is purposeful to a very select group of consumers; mostly spontaneous dancers, street performers, or perhaps ventriloquists benefit immediately from our invention. With more selective representations of common movements and smoother transitions between song adjustments, the Dynamic DJ can be suitable for any mood or occasion.

The external hardware is currently not practical for daily outdoor use or excessive or straining movements. We’d like to increase its flexibility via thinner wires such as wire thread or wireless communication via Bluetooth.

# Future Ideas

We managed to input commands using the orientations of the accelerometers and manipulate music based on these commands. This was a success, though somewhat simpler than our original goals and ideas. Instead of putting in many songs that have similar genres and time periods and have the Arduino play one at random, we instead assigned one song to be played when a certain song or decade was selected. Given more time, we would find out why our original code for this would not work.

A similar issue we had was getting the skip and speed up/slow down functions to work. The speed controls could only speed the song up, but not slow it down, so we had to switch them out for the bass and treble controls. The skip functions did not work altogether, and so songs had to be stopped, selected and then played in order to change songs.

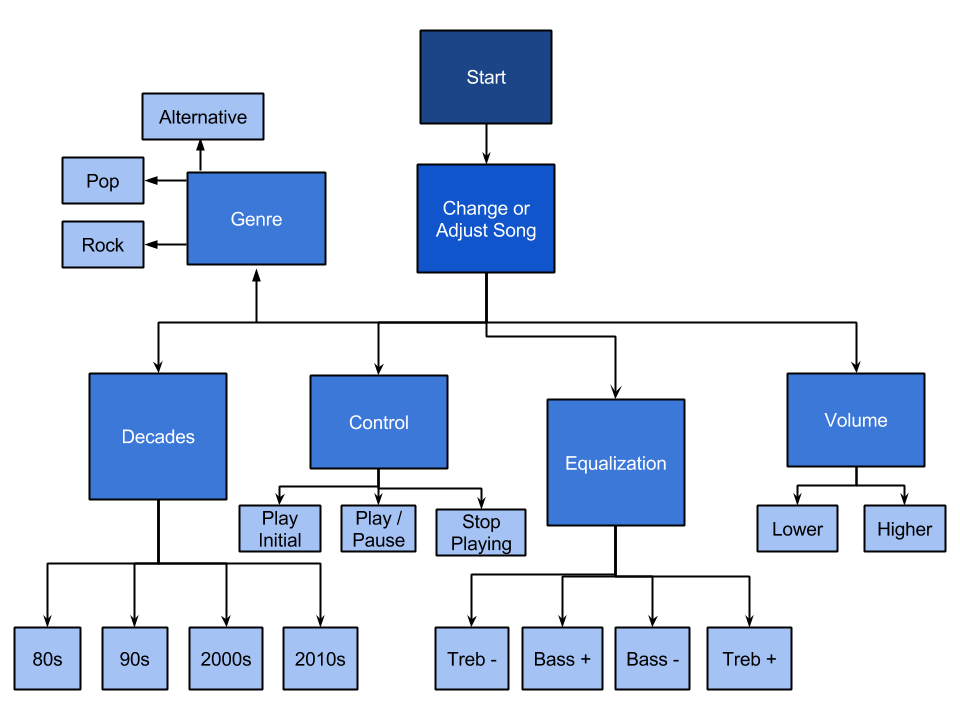
We would also like to interpret more movements than the 25 orientations we had. This could range anywhere from adding rate gyros, to interpreting accelerations from these orientations, to recognizing specific hand movements based on a combination of orientation and acceleration. This could lead to a much more complex and intricate user interface that would give greater control over the music.

Due to time constraints, we were unable to implement a switch that allowed a user to go from “control mode” to “dance mode.” We would also have liked to implement certain mechanical upgrades, including more flexible wire and a harness that is easier to put on and safer for the electrical components (i.e. less risk for wires or resistors coming unplugged, etc.).

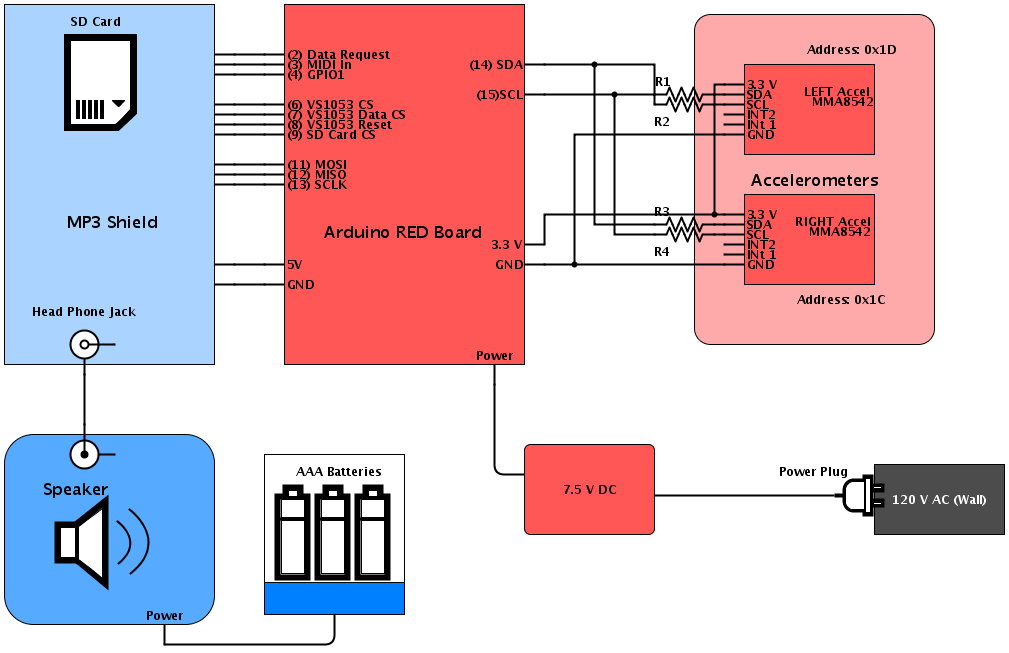
# Appendices

## Appendix A

## Appendix B



## Appendix C



## Appendix D

#include <SPI.h>

#include <SdFat.h>

#include <SdFatUtil.h>

#include <SFEMP3Shield.h>

#include <Wire.h>

#include <SFE\_MMA8452Q.h>

//declaring accelerometers

MMA8452Q Aleft;

MMA8452Q Aright=MMA8452Q(0x1C);

#if defined(USE\_MP3\_REFILL\_MEANS) && USE\_MP3\_REFILL\_MEANS == USE\_MP3\_Timer1

#include <TimerOne.h>

#elif defined(USE\_MP3\_REFILL\_MEANS) && USE\_MP3\_REFILL\_MEANS == USE\_MP3\_SimpleTimer

#include <SimpleTimer.h>

#endif

SdFat sd;

SFEMP3Shield MP3player;

int16\_t last\_ms\_char; // milliseconds of last recieved character from Serial port.

int8\_t buffer\_pos; // next position to recieve character from Serial port.

char buffer[6]; // 0-35K+null

char stateLeft;

char stateRight;

boolean flatVol=false;

boolean flatTempo=false;

boolean choosepps=true;

boolean choseDecade=true;

void setup() {

uint8\_t result; //result code from some function as to be tested at later time.

Serial.begin(115200);

Serial.println("In setup");

Aleft.init();

Serial.println(" past left");

Aright.init();

Serial.println("leaving Flor's setup");

Serial.print(F("F\_CPU = "));

Serial.println(F\_CPU);

Serial.print(F("Free RAM = ")); // available in Version 1.0 F() bases the string to into Flash, to use less SRAM.

Serial.print(FreeRam(), DEC); // FreeRam() is provided by SdFatUtil.h

Serial.println(F(" Should be a base line of 1017, on ATmega328 when using INTx"));

//Initialize the SdCard.

if(!sd.begin(SD\_SEL, SPI\_FULL\_SPEED)) sd.initErrorHalt();

// depending upon your SdCard environment, SPI\_HAVE\_SPEED may work better.

if(!sd.chdir("/")) sd.errorHalt("sd.chdir");

//Initialize the MP3 Player Shield

result = MP3player.begin();

//check result, see readme for error codes.

if(result != 0) {

Serial.print(F("Error code: "));

Serial.print(result);

Serial.println(F(" when trying to start MP3 player"));

if( result == 6 ) {

Serial.println(F("Warning: patch file not found, skipping.")); // can be removed for space, if needed.

Serial.println(F("Use the \"d\" command to verify SdCard can be read")); // can be removed for space, if needed.

}

}

#if (0)

// Typically not used by most shields, hence commented out.

Serial.println(F("Applying ADMixer patch."));

if(MP3player.ADMixerLoad("admxster.053") == 0) {

Serial.println(F("Setting ADMixer Volume."));

MP3player.ADMixerVol(-3);

}

#endif

}

//------------------------------------------------------------------------------

/\*\*

\* \brief Main Loop the Arduino Chip

\*

\* This is called at the end of Arduino kernel's main loop before recycling.

\* And is where the user's serial input of bytes are read and analyzed by

\* parsed\_menu.

\*

\* Additionally, if the means of refilling is not interrupt based then the

\* MP3player object is serviced with the availaible function.

\*

\* \note Actual examples of the libraries public functions are implemented in

\* the parse\_menu() function.

\*/

void loop() {

// Below is only needed if not interrupt driven. Safe to remove if not using.

#if defined(USE\_MP3\_REFILL\_MEANS) \

&& ( (USE\_MP3\_REFILL\_MEANS == USE\_MP3\_SimpleTimer) \

|| (USE\_MP3\_REFILL\_MEANS == USE\_MP3\_Polled) )

MP3player.available();

#endif

char stateLeft;

char stateRight;

boolean flatVol=false;

boolean flatTempo=false;

boolean choosepps=true;

boolean choseDecade=true;

if( Aleft.available()&& Aright.available()) {

//Serial.println("reading accel");

Aleft.read();

Aright.read();

stateLeft=OrientationLEFT();

stateRight=OrientationRIGHT();

//Serial.println(stateLEFT);

//Serial.print(stateRIGHT);

parse\_menu(stateLeft, stateRight);

}

delay(100);

}

uint32\_t millis\_prv;

//------------------------------------------------------------------------------

/\*\*

\* \brief Decode the Menu.

\*

\* Parses through the characters of the users input, executing corresponding

\* MP3player library functions and features then displaying a brief menu and

\* prompting for next input command.

\*/

void parse\_menu(char stateLeft, char stateRight) {

int i=3;

uint8\_t result; // result code from some function as to be tested at later time.

// Note these buffer may be desired to exist globably.

// but do take much space if only needed temporarily, hence they are here.

char title[30]; // buffer to contain the extract the Title from the current filehandles

char artist[30]; // buffer to contain the extract the artist name from the current filehandles

char album[30]; // buffer to contain the extract the album name from the current filehandles

char year[4];

/\* VOLUME CONTROL \*/

if(stateLeft == 'D') {

union twobyte mp3\_vol; // create key\_command existing variable that can be both word and double byte of left and right.

mp3\_vol.word = MP3player.getVolume(); // returns a double uint8\_t of Left and Right packed into int16\_t

if (stateRight == 'F') {

flatVol=true;

}

if(stateRight == 'D' && (flatVol)) { // note dB is negative

// assume equal balance and use byte[1] for math

if(mp3\_vol.byte[1] >= 250) { // range check

mp3\_vol.byte[1] = 250;

} else {

mp3\_vol.byte[1] += 6; // keep it simpler with whole dB's

}

// push byte[1] into both left and right assuming equal balance.

MP3player.setVolume(mp3\_vol.byte[1], mp3\_vol.byte[1]); // commit new volume

Serial.print(F("Volume changed to -"));

Serial.print(mp3\_vol.byte[1]>>1, 1);

Serial.println(F("[dB]"));

flatVol=false;

}

if(stateRight == 'U' && (flatVol)) {

if(mp3\_vol.byte[1] <= 6) { // range check

mp3\_vol.byte[1] = 6;

} else {

mp3\_vol.byte[1] -= 6;

}

// push byte[1] into both left and right assuming equal balance.

MP3player.setVolume(mp3\_vol.byte[1], mp3\_vol.byte[1]); // commit new volume

Serial.print(F("Volume changed to -"));

Serial.print(mp3\_vol.byte[1]>>1, 1);

Serial.println(F("[dB]"));

flatVol=false;

}

if (stateRight == 'R' && (flatVol)) {

byte key\_command;

key\_command =( i+'0') - 48;

#if USE\_MULTIPLE\_CARDS

sd.chvol(); // assign desired sdcard's volume.

#endif

//tell the MP3 Shield to play a track

result = MP3player.playTrack(key\_command);

//check result, see readme for error codes.

if(result != 0) {

Serial.print(F("Error code: "));

Serial.print(result);

Serial.println(F(" when trying to play track"));

} else {

Serial.println(F("Playing:"));

//we can get track info by using the following functions and arguments

//the functions will extract the requested information, and put it in the array we pass in

MP3player.trackTitle((char\*)&title);

MP3player.trackArtist((char\*)&artist);

MP3player.trackAlbum((char\*)&album);

MP3player.trackYear((char\*)&year);

//print out the arrays of track information

Serial.write((byte\*)&title, 30);

Serial.println();

Serial.print(F("by: "));

Serial.write((byte\*)&artist, 30);

Serial.println();

Serial.print(F("Album: "));

Serial.write((byte\*)&album, 30);

Serial.println();

Serial.print(F("Year: "));

Serial.write((byte\*)&year, 4);

Serial.println();

}

flatVol=false;

}

if (stateRight == 'L' && (flatVol)) {

if( MP3player.getState() == playback) {

MP3player.pauseMusic();

Serial.println(F("Pausing"));

} else if( MP3player.getState() == paused\_playback) {

MP3player.resumeMusic();

Serial.println(F("Resuming"));

} else {

Serial.println(F("Not Playing!"));

}

flatVol=false;

}

/\* Genre/Stop \*/

} else if(stateLeft == 'U') {

if(stateRight == 'F') {

choosepps=true;

}

if(stateRight == 'U' && choosepps) {

byte key\_command;

key\_command =('6' - 48);

#if USE\_MULTIPLE\_CARDS

sd.chvol(); // assign desired sdcard's volume.

#endif

//tell the MP3 Shield to play a track

result = MP3player.playTrack(key\_command);

//check result, see readme for error codes.

if(result != 0) {

Serial.print(F("Error code: "));

Serial.print(result);

Serial.println(F(" when trying to play track"));

} else {

Serial.println(F("Playing:"));

//we can get track info by using the following functions and arguments

//the functions will extract the requested information, and put it in the array we pass in

MP3player.trackTitle((char\*)&title);

MP3player.trackArtist((char\*)&artist);

MP3player.trackAlbum((char\*)&album);

MP3player.trackYear((char\*)&year);

//print out the arrays of track information

Serial.write((byte\*)&title, 30);

Serial.println();

Serial.print(F("by: "));

Serial.write((byte\*)&artist, 30);

Serial.println();

Serial.print(F("Album: "));

Serial.write((byte\*)&album, 30);

Serial.println();

Serial.print(F("Year: "));

Serial.write((byte\*)&year, 4);

Serial.println();

}

choosepps=false;

}

if(stateRight == 'R' && choosepps) {

byte key\_command;

key\_command = '1' - 48;

#if USE\_MULTIPLE\_CARDS

sd.chvol(); // assign desired sdcard's volume.

#endif

//tell the MP3 Shield to play a track

result = MP3player.playTrack(key\_command);

//check result, see readme for error codes.

if(result != 0) {

Serial.print(F("Error code: "));

Serial.print(result);

Serial.println(F(" when trying to play track"));

} else {

Serial.println(F("Playing:"));

//we can get track info by using the following functions and arguments

//the functions will extract the requested information, and put it in the array we pass in

MP3player.trackTitle((char\*)&title);

MP3player.trackArtist((char\*)&artist);

MP3player.trackAlbum((char\*)&album);

MP3player.trackYear((char\*)&year);

//print out the arrays of track information

Serial.write((byte\*)&title, 30);

Serial.println();

Serial.print(F("by: "));

Serial.write((byte\*)&artist, 30);

Serial.println();

Serial.print(F("Album: "));

Serial.write((byte\*)&album, 30);

Serial.println();

Serial.print(F("Year: "));

Serial.write((byte\*)&year, 4);

Serial.println();

}

choosepps=false;

}

if(stateRight == 'L' && choosepps) {

MP3player.stopTrack();

choosepps=false;

}

if (stateRight == 'D' && (choosepps)) {

byte key\_command;

key\_command ='4' - 48;

#if USE\_MULTIPLE\_CARDS

sd.chvol(); // assign desired sdcard's volume.

#endif

//tell the MP3 Shield to play a track

result = MP3player.playTrack(key\_command);

//check result, see readme for error codes.

if(result != 0) {

Serial.print(F("Error code: "));

Serial.print(result);

Serial.println(F(" when trying to play track"));

} else {

Serial.println(F("Playing:"));

//we can get track info by using the following functions and arguments

//the functions will extract the requested information, and put it in the array we pass in

MP3player.trackTitle((char\*)&title);

MP3player.trackArtist((char\*)&artist);

MP3player.trackAlbum((char\*)&album);

MP3player.trackYear((char\*)&year);

//print out the arrays of track information

Serial.write((byte\*)&title, 30);

Serial.println();

Serial.print(F("by: "));

Serial.write((byte\*)&artist, 30);

Serial.println();

Serial.print(F("Album: "));

Serial.write((byte\*)&album, 30);

Serial.println();

Serial.print(F("Year: "));

Serial.write((byte\*)&year, 4);

Serial.println();

}

choosepps=false;

}

} else if(stateLeft == 'R') {

if(stateRight == 'F') {

flatTempo=true;

}

if (stateRight == 'R' && (flatTempo)) {

int8\_t TrebleAmplitude = MP3player.getTrebleAmplitude();

Serial.print(F("Former TrebleAmplitude = "));

Serial.println(TrebleAmplitude, DEC);

if (TrebleAmplitude >= 7) { // Range is from -8 - 7dB

TrebleAmplitude = -8;

} else {

TrebleAmplitude+=2;

}

MP3player.setTrebleAmplitude(TrebleAmplitude);

Serial.print(F("New TrebleAmplitude = "));

Serial.println(MP3player.getTrebleAmplitude(), DEC);

flatTempo=false;

}

if (stateRight == 'L' && (flatTempo)) {

int8\_t TrebleAmplitude = MP3player.getTrebleAmplitude();

Serial.print(F("Former TrebleAmplitude = "));

Serial.println(TrebleAmplitude, DEC);

if (TrebleAmplitude <= -8) { // Range is from -8 - 7dB

TrebleAmplitude = 7;

} else {

TrebleAmplitude-=2;

}

MP3player.setTrebleAmplitude(TrebleAmplitude);

Serial.print(F("New TrebleAmplitude = "));

Serial.println(MP3player.getTrebleAmplitude(), DEC);

flatTempo=false;

}

if (stateRight == 'U' && (flatTempo)) {

uint16\_t BassAmplitude = MP3player.getBassAmplitude();

Serial.print(F("Former BassAmplitude = "));

Serial.println(BassAmplitude, DEC);

if (BassAmplitude >= 15) { // Range is from 0 - 15dB

BassAmplitude = 0;

} else {

BassAmplitude+=2;

}

MP3player.setBassAmplitude(BassAmplitude);

Serial.print(F("New BassAmplitude = "));

Serial.println(MP3player.getBassAmplitude(), DEC);

flatTempo=false;

}

if (stateRight == 'D' && (flatTempo)) {

uint16\_t BassAmplitude = MP3player.getBassAmplitude();

Serial.print(F("Former BassAmplitude = "));

Serial.println(BassAmplitude, DEC);

if (BassAmplitude <= 0) { // Range is from 0 - 15dB

BassAmplitude = 15;

} else {

BassAmplitude-=2;

}

MP3player.setBassAmplitude(BassAmplitude);

Serial.print(F("New BassAmplitude = "));

Serial.println(MP3player.getBassAmplitude(), DEC);

flatTempo=false;

}

} else if(stateLeft == 'L') {

if(stateRight == 'F') {

choseDecade=true;

}

if(stateRight == 'U' && (choseDecade)) {

//2010+

char ans;

find2010((char\*) &ans);

if(ans!='n') {

byte key\_command =ans - 48;

#if USE\_MULTIPLE\_CARDS

sd.chvol(); // assign desired sdcard's volume.

#endif

//tell the MP3 Shield to play a track

result = MP3player.playTrack(key\_command);

//check result, see readme for error codes.

if(result != 0) {

Serial.print(F("Error code: "));

Serial.print(result);

Serial.println(F(" when trying to play track"));

} else {

Serial.println(F("Playing:"));

//we can get track info by using the following functions and arguments

//the functions will extract the requested information, and put it in the array we pass in

MP3player.trackTitle((char\*)&title);

MP3player.trackArtist((char\*)&artist);

MP3player.trackAlbum((char\*)&album);

MP3player.trackYear((char\*)&year);

//print out the arrays of track information

Serial.write((byte\*)&title, 30);

Serial.println();

Serial.print(F("by: "));

Serial.write((byte\*)&artist, 30);

Serial.println();

Serial.print(F("Album: "));

Serial.write((byte\*)&album, 30);

Serial.println();

Serial.print(F("Year: "));

Serial.write((byte\*)&year, 4);

Serial.println();

}

choseDecade=false;

}

}

if(stateRight == 'R' && (choseDecade)) {

//2000-2010

char ans;

find2000((char\*) &ans);

if(ans!='n') {

byte key\_command =ans - 48;

#if USE\_MULTIPLE\_CARDS

sd.chvol(); // assign desired sdcard's volume.

#endif

//tell the MP3 Shield to play a track

result = MP3player.playTrack(key\_command);

//check result, see readme for error codes.

if(result != 0) {

Serial.print(F("Error code: "));

Serial.print(result);

Serial.println(F(" when trying to play track"));

} else {

Serial.println(F("Playing:"));

//we can get track info by using the following functions and arguments

//the functions will extract the requested information, and put it in the array we pass in

MP3player.trackTitle((char\*)&title);

MP3player.trackArtist((char\*)&artist);

MP3player.trackAlbum((char\*)&album);

MP3player.trackYear((char\*)&year);

//print out the arrays of track information

Serial.write((byte\*)&title, 30);

Serial.println();

Serial.print(F("by: "));

Serial.write((byte\*)&artist, 30);

Serial.println();

Serial.print(F("Album: "));

Serial.write((byte\*)&album, 30);

Serial.println();

Serial.print(F("Year: "));

Serial.write((byte\*)&year, 4);

Serial.println();

}

choseDecade=false;

}

}

if(stateRight == 'D' && (choseDecade)) {

//1990-2000

char ans;

find1990((char\*) &ans);

if(ans!='n') {

byte key\_command =ans - 48;

#if USE\_MULTIPLE\_CARDS

sd.chvol(); // assign desired sdcard's volume.

#endif

//tell the MP3 Shield to play a track

result = MP3player.playTrack(key\_command);

//check result, see readme for error codes.

if(result != 0) {

Serial.print(F("Error code: "));

Serial.print(result);

Serial.println(F(" when trying to play track"));

} else {

Serial.println(F("Playing:"));

//we can get track info by using the following functions and arguments

//the functions will extract the requested information, and put it in the array we pass in

MP3player.trackTitle((char\*)&title);

MP3player.trackArtist((char\*)&artist);

MP3player.trackAlbum((char\*)&album);

MP3player.trackYear((char\*)&year);

//print out the arrays of track information

Serial.write((byte\*)&title, 30);

Serial.println();

Serial.print(F("by: "));

Serial.write((byte\*)&artist, 30);

Serial.println();

Serial.print(F("Album: "));

Serial.write((byte\*)&album, 30);

Serial.println();

Serial.print(F("Year: "));

Serial.write((byte\*)&year, 4);

Serial.println();

}

choseDecade=false;

}

}

if(stateRight == 'L' && (choseDecade)) {

//1990-

char ans;

find1980((char\*) &ans);

if(ans!='n') {

byte key\_command =ans - 48;

#if USE\_MULTIPLE\_CARDS

sd.chvol(); // assign desired sdcard's volume.

#endif

//tell the MP3 Shield to play a track

result = MP3player.playTrack(key\_command);

//check result, see readme for error codes.

if(result != 0) {

Serial.print(F("Error code: "));

Serial.print(result);

Serial.println(F(" when trying to play track"));

} else {

Serial.println(F("Playing:"));

//we can get track info by using the following functions and arguments

//the functions will extract the requested information, and put it in the array we pass in

MP3player.trackTitle((char\*)&title);

MP3player.trackArtist((char\*)&artist);

MP3player.trackAlbum((char\*)&album);

MP3player.trackYear((char\*)&year);

//print out the arrays of track information

Serial.write((byte\*)&title, 30);

Serial.println();

Serial.print(F("by: "));

Serial.write((byte\*)&artist, 30);

Serial.println();

Serial.print(F("Album: "));

Serial.write((byte\*)&album, 30);

Serial.println();

Serial.print(F("Year: "));

Serial.write((byte\*)&year, 4);

Serial.println();

}

choseDecade=false;

}

}

}

}

char OrientationLEFT()

{

// accel.readPL() will return a byte containing information

// about the orientation of the sensor. It will be either

// PORTRAIT\_U, PORTRAIT\_D, LANDSCAPE\_R, LANDSCAPE\_L, or

// LOCKOUT.

byte pl = Aleft.readPL();

switch (pl)

{

case PORTRAIT\_U:

return('U');

break;

case PORTRAIT\_D:

return('D');

break;

case LANDSCAPE\_R:

return('R');

break;

case LANDSCAPE\_L:

return('L');

break;

case LOCKOUT:

return('F');

break;

}

}

char OrientationRIGHT() {

// accel.readPL() will return a byte containing information

// about the orientation of the sensor. It will be either

// PORTRAIT\_U, PORTRAIT\_D, LANDSCAPE\_R, LANDSCAPE\_L, or

// LOCKOUT.

byte pl = Aright.readPL();

switch (pl)

{

case PORTRAIT\_U:

return('U');

break;

case PORTRAIT\_D:

return('D');

break;

case LANDSCAPE\_R:

return('R');

break;

case LANDSCAPE\_L:

return('L');

break;

case LOCKOUT:

return('F');

break;

}

}

void find2010(char\* information) {

char hey;

char year[4];

MP3player.playTrack('1'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if(year>="2010") {

hey='1';

}

else {

hey='n';

}

MP3player.playTrack('2'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if(year>="2010") {

hey='2';

}

else {

hey='n';

}

MP3player.playTrack('3'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if(year>="2010") {

hey='3';

}

else {

hey='n';

}

MP3player.playTrack('4'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if(year>="2010") {

hey='4';

}

else {

hey='n';

}

MP3player.playTrack('5'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if(year>="2010") {

hey='5';

}

else {

hey='n';

}

MP3player.playTrack('6'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if(year>="2010") {

hey='6';

}

else {

hey='n';

}

MP3player.playTrack('7'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if(year>="2010") {

hey='7';

}

else {

hey='n';

}

MP3player.playTrack('8'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if(year>="2010") {

hey='8';

}

else {

hey='n';

}

MP3player.playTrack('9'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if(year>="2010") {

hey='9';

}

else {

hey='n';

}

\*information=hey;

}

void find2000(char\* information) {

char hey;

char year[4];

MP3player.playTrack('1'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if((year<="2009") && (year>="2000")) {

hey='1';

}

else {

hey='n';

}

MP3player.playTrack('2'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if((year<="2009") && (year>="2000")) {

hey='2';

}

else {

hey='n';

}

MP3player.playTrack('3'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if((year<="2009") && (year>="2000")) {

hey='3';

}

else {

hey='n';

}

MP3player.playTrack('4'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if((year<="2009") && (year>="2000")) {

hey='4';

}

else {

hey='n';

}

MP3player.playTrack('5'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if((year<="2009") && (year>="2000")) {

hey='5';

}

else {

hey='n';

}

MP3player.playTrack('6'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if((year<="2009") && (year>="2000")) {

hey='6';

}

else {

hey='n';

}

MP3player.playTrack('7'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if((year<="2009") && (year>="2000")) {

hey='7';

}

else {

hey='n';

}

MP3player.playTrack('8'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if((year<="2009") && (year>="2000")) {

hey='8';

}

else {

hey='n';

}

MP3player.playTrack('9'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if((year<="2009") && (year>="2000")) {

hey='9';

}

else {

hey='n';

}

\*information=hey;

}

void find1990(char\* information) {

char hey;

char year[4];

MP3player.playTrack('1'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if((year<="1999") && (year>="1990")) {

hey='1';

}

else {

hey='n';

}

MP3player.playTrack('2'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if((year<="1999") && (year>="1990")) {

hey='2';

}

else {

hey='n';

}

MP3player.playTrack('3'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if((year<="1999") && (year>="1990")) {

hey='3';

}

else {

hey='n';

}

MP3player.playTrack('4'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if((year<="1999") && (year>="1990")) {

hey='4';

}

else {

hey='n';

}

MP3player.playTrack('5'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if((year<="1999") && (year>="1990")) {

hey='5';

}

else {

hey='n';

}

MP3player.playTrack('6'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if((year<="1999") && (year>="1990")) {

hey='6';

}

else {

hey='n';

}

MP3player.playTrack('7'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if((year<="1999") && (year>="1990")) {

hey='7';

}

else {

hey='n';

}

MP3player.playTrack('8'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if((year<="1999") && (year>="1990")) {

hey='8';

}

else {

hey='n';

}

MP3player.playTrack('9'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if((year<="1999") && (year>="1990")) {

hey='9';

}

else {

hey='n';

}

\*information=hey;

}

void find1980(char\* information) {

char hey;

char year[4];

MP3player.playTrack('1'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if(year<="1989") {

hey='1';

}

else {

hey='n';

}

MP3player.playTrack('2'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if(year<="1989") {

hey='2';

}

else {

hey='n';

}

MP3player.playTrack('3'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if(year<="1989") {

hey='3';

}

else {

hey='n';

}

MP3player.playTrack('4'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if(year<="1989") {

hey='4';

}

else {

hey='n';

}

MP3player.playTrack('5'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if(year<="1989") {

hey='5';

}

else {

hey='n';

}

MP3player.playTrack('6'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if(year<="1989") {

hey='6';

}

else {

hey='n';

}

MP3player.playTrack('7'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if(year<="1989") {

hey='7';

}

else {

hey='n';

}

MP3player.playTrack('8'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if(year<="1989") {

hey='8';

}

else {

hey='n';

}

MP3player.playTrack('9'-48);

delay(10);

MP3player.pauseMusic();

MP3player.trackYear((char\*)&year);

if(year<="1989") {

hey='9';

}

else {

hey='n';

}

\*information=hey;

}