

[arduino](#), [lecteur-son](#), [em](#)

Téléphone BMR

/!\ en cours de rédaction 22 octobre 2019

C'est un superbe téléphone sensuel en forme de bouche qui a servi à diffuser une suite d'entretiens sur le sujet des règles. Il a été utilisé comme borne d'écoute pour la journée/soirée "Bois mes règles" organisée par Gast! à Quimper / La Baleine en mars 2018.

Le clavier du téléphone permet de déclencher la lecture d'un enregistrement que l'on peut écouter dans le combiné, comme un téléphone quoi!

Le circuit du téléphone est remplacé par un arduino nano qui déclenche la lecture d'un fichier mp3 par un circuit DFPlayer, le haut-parleur utilisé est le haut-parleur original du combiné. Le clavier original du téléphone est aussi réutilisé et l'ensemble est alimenté par un transfo qui récupère le courant du secteur pour en faire un courant 9V continu.

Réutiliser le haut parleur du téléphone

Le combiné est relié au poste par une prise [RJ11](#) (seuls quatre fils sont utilisés) :

- Vert : micro
- Rouge : micro
- Jaune : HP
- Noir : HP

On se servira des fils jaunes et noirs pour relier le haut-parleur au circuit

Réutiliser le clavier du téléphone

Chaque touche active une liaison entre deux broches.

Méthode pour retrouver la matrice :

- numéroter les broches ou les nommer
- sur du papier, dessiner une grille avec autant de colonnes et de rangées que sur le clavier et dessiner le clavier
- en utilisant un multimètre pour tester la continuité, tester toutes les combinaisons de broches possibles, en appuyant sur chaque touche, l'une après l'autre
- noter pour chaque touche les 2 broches utilisées

Connexions du clavier sur nappe : R4, R3, R2, R1, VSS, C1, C2, C3, C4, PT

| | C1 | C2 | C3 | C4 |
|----|----|----|----|--------|
| R1 | 1 | 2 | 3 | |
| R2 | 4 | 5 | 6 | FLASH |
| R3 | 7 | 8 | 9 | |
| R4 | * | 0 | # | REDIAL |



telephone_BMR_test_clavier_002.ino (cliquer pour afficher le code)

[telephone_BMR_test_clavier_002.ino](#)

```

/* deuxième essai avec une autre library pour l'afficheur 7 segment
 * https://github.com/bremme/arduino-tm1637
 * SevenSegmentTM1637 de Bram Harmsen
 */

// *****
// Définitions pour l'afficheur U7 segments à 4 chiffres
#include "SevenSegmentTM1637.h"

const byte PIN_CLK = 2; // define CLK pin (any digital pin)
const byte PIN_DIO = 3; // define DIO pin (any digital pin)
SevenSegmentTM1637 display(PIN_CLK, PIN_DIO);

// *****
// Définitions pour le clavier matrice
#include <Keypad.h>

const byte ROWS = 4; // quatre rangées
const byte COLS = 4; // quatre colonnes
char keys[ROWS][COLS] = {
  {'1','2','3','Z'},
  {'4','5','6','F'},
  {'7','8','9','Z'},
  {'*','0','#','R'}
};

/* Correspondances entre les broches identifiées sur le téléphone et les pins de l'arduino
   R1 : 7      C1 : 8
   R2 : 6      C2 : 9
   R3 : 5      C3 : 10
   R4 : 4      C4 : 11      */
byte rowPins[ROWS] = { 7, 6, 5, 4}; // connecter les rangées à ces pins
byte colPins[COLS] = { 8, 9, 10, 11}; // connecter les colonnes à ces pins

Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

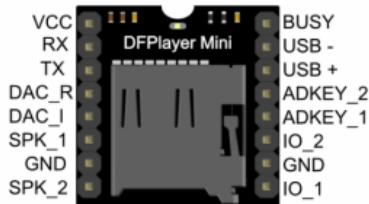
void setup() {
  Serial.begin(9600);
  // Initialiser l'afficheur à 4 chiffres
  display.begin(); // initializes the display
  display.setBacklight(100); // set the brightness to 100 %
  display.print("INIT"); // display INIT on the display
  delay(1000); // wait 1000 ms
}

void loop() {
  char key = keypad.getKey();

  if (key){
    display.print(key);
    Serial.println(key);
    delay(300);
  } else {
    display.clear();
  }
}

```

Utiliser un DFPlayer mini



Pour utiliser le DFPlayer mini je me suis basé sur la documentation du fabricant du module (DFRobot), mais j'ai pu lire des commentaires sur cette doc. qui disaient qu'elle est incomplète, et Ype Brada a proposé le code ci-dessous pour utiliser toutes les fonctions du DFPlayer mini. Je me suis aussi rendu compte qu'il fallait utiliser `myDFPlayer.enableDAC()` ; pour permettre la sortie sur le DAC (pour ampli, casque, etc.)

DFPlayer_Command_Discovery.ino (cliquer pour afficher le code)

[DFPlayer_Command_Discovery.ino](#)

```
/*
DF Player mini command discovery
-----

Program is meant to discover all the possibilities of the command
structure of the DFPlayer mini. No special libraries are needed.

In general, there are 3 ways to address a MP3 or WAV file:
(1) Track order:
    All songs are stored in a certain order on the card. The order is however
    not guaranteed and very dependent on the order how the files were written to the card. This
    method is not suitable when it is absolutely necessary that a specific track is played. Commands
    using this written order are (0x01, 0x02, 0x03, 0x08, 0x11, 0x3C, 0x3D, 0x3E, 0x4B, 0x4C, 0x4D).
    The name of a song is arbitrary.
(2) Folder number and song number:
    Folders are named 01-99 and songs 001-255.mp3 or 001-255.WAV. It's possible to exactly address
    a specific song with command 0x0F. Command 0x17 is related. When addressing a specific file,
    the file is always internally converted to the stored track number. This number can be requested by
    commands 0x4B, 0x4C or 0x4D.
(3) Folder named "mp3" and song number:
    A folder is named "mp3" and songs with a name of exact 4-digit number (0001-2999) e.g. 0235.mp3.
    It's possible to exactly address a specific song with command 0x12. The according track number can
    be requested by commands 0x4B, 0x4C or 0x4D.

This program is simple and can be the basis for your own mp3 player sketch.
Note: The DF Player commands are not always correctly described in the manual. I tried to fix it, but
there is still something to do. The commands recovered so far are listed below.

How to use this sketch:
Enter three (separated) DECIMAL numbers in the Serial Monitor with no end of line character.
First number : Command
Second number: First (High Byte) parameter
Third number : Second (Low Byte) parameter
E.g.: 3,0,1 will play the first track on the TF card

VERY IMPORTANT: Use serial 1K resistors or a level shifter between module RX and TX
and Arduino to suppress audio noise
Connect Sound module board RX to Arduino pin 11 (via 1K resistor)
Connect Sound module board TX to Arduino pin 10 (via 1K resistor)
Connect Sound module board Vcc to Arduino Vin when powered via USB (preferably 3.0)
else use separate 5V power supply
Connect Sound module board GND to Arduino GND

General DF Player mini command structure (only byte 3, 5 and 6 to be entered in the serial monitor):
-----
Byte Function      Value
=====
(0) Start Byte    0x7E
(1) Version Info  0xFF (don't know why it's called Version Info)
(2) Number of bytes 0x06 (Always 6 bytes)
(3) Command        0x__
(4) Command feedback 0x__ If enabled returns info with command 0x41 [0x01: info, 0x00: no info]
(5) Parameter 1 [DH] 0x__
(6) Parameter 2 [DL] 0x__
(7) Checksum high  0x__ See explanation below. Is calculated in function: execute_CMD
(8) Checksum low   0x__ See explanation below. Is calculated in function: execute_CMD
(9) End command    0xEF

Checksum calculation.
-----
Checksum = -Sum(byte(1..6)) (2 bytes, notice minus sign!)
```

DF Player mini Commands without returned parameters (*=Confirmed command ?=Unknown, not clear or not validated)

| CMD | CMD | HEX | DEC | Function Description | Parameters(2 x 8 bit) |
|------|-----|-------------------------------------|-----|--|-----------------------|
| 0x01 | 1 | Next | * | [DH]=X, [DL]=X Next track in current folder.Loops when last file played | |
| 0x02 | 2 | Previous | * | [DH]=X, [DL]=X Previous track in current folder.Loops when last file played | |
| 0x03 | 3 | Specify track(NUM) | * | [DH]=highByte(NUM), [DL]=lowByte(NUM) 1-2999 Playing order is order in which the numbers are stored. Filename and foldername are arbitrary, but when named starting with an increasing number and then placed in one folder, files are (often) played in that order and with correct track number. e.g. 0001-Joe Jackson.mp3...0348-Lets dance.mp3) | |
| 0x04 | 4 | Increase volume | * | [DH]=X, [DL]=X Increase volume by 1 | |
| 0x05 | 5 | Decrease volume | * | [DH]=X, [DL]=X Decrease volume by 1 | |
| 0x06 | 6 | Specify volume | * | [DH]=X, [DL]= Volume (0-0x30) Default=0x30 | |
| 0x07 | 7 | Specify Equalizer | * | [DH]=X, [DL]= EQ(0/1/2/3/4/5) [Normal/Pop/Rock/Jazz/Classic/Base] | |
| 0x08 | 8 | Specify repeat(NUM) | * | [DH]=highByte(NUM), [DL]=lowByte(NUM).Repeat the specified track number | |
| 0x09 | 9 | Specify playback source (Datasheet) | ? | [DH]=X, [DL]= (0/1/2/3/4)Unknown. Seems to be overridden by automatic detection (Datasheet: U/TF/AUX/SLEEP/FLASH) | |
| 0x0A | 10 | Enter into standby - low power loss | * | [DH]=X, [DL]=X Works, but no command found yet to end standby (insert TF-card again will end standby mode) | |
| 0x0B | 11 | Normal working (Datasheet) | ? | Unknown. No error code, but no function found | |
| 0x0C | 12 | Reset module | * | [DH]=X, [DL]=X Resets all (Track = 0x01, Volume = 0x30) Will return 0x3F initialization parameter (0x02 for TF-card) "Clap" sound after executing command (no solution found) | |
| 0x0D | 13 | Play | * | [DH]=X, [DL]=X Play pointered track | |
| 0x0E | 14 | Pause | * | [DH]=X, [DL]=X Pause track | |
| 0x0F | 15 | Specify folder and file to playback | * | [DH]=Folder, [DL]=File Important: Folders must be named 01-99, files must be named 001-255 | |
| 0x10 | 16 | Volume adjust set (Datasheet) | ? | Unknown. No error code. Does not change the volume gain. | |
| 0x11 | 17 | Loop play | * | [DH]=X, [DL]=(0x01:play, 0x00:stop play) Loop play all the tracks. Start at track 1. | |
| 0x12 | 18 | Play mp3 file [NUM] in mp3 folder | * | [DH]=highByte(NUM), [DL]=lowByte(NUM) Play mp3 file in folder named mp3 in your TF-card. File format exact 4-digit number (0001-2999) e.g. 0235.mp3 | |
| 0x13 | 19 | Unknown | ? | Unknown: Returns error code 0x07 | |
| 0x14 | 20 | Unknown | ? | Unknown: Returns error code 0x06 | |
| 0x15 | 21 | Unknown | ? | Unknown: Returns no error code, but no function found | |
| 0x16 | 22 | Stop | * | [DH]=X, [DL]=X, Stop playing current track | |
| 0x17 | 23 | Loop Folder "01" | * | [DH]=x, [DL]=1-255, Loops all files in folder named "01" | |
| 0x18 | 24 | Random play | * | [DH]=X, [DL]=X Random all tracks, always starts at track 1 | |
| 0x19 | 25 | Single loop | * | [DH]=0, [DL]=0 Loops the track that is playing | |
| 0x1A | 26 | Pause | * | [DH]=X, [DL]=(0x01:pause, 0x00:stop pause) | |

Commands with returned parameters (*=Confirmed command ?=Unknown, not clear or not validated)

| CMD | CMD | HEX | DEC | Function Description | Parameters(2 x 8 bit) |
|------|-----|-----------------------------------|-----|--|-----------------------|
| 0x3A | 58 | Medium inserted | * | [DH]=0, [DL]=(1:U-disk, 2:TF-card) | |
| 0x3B | 59 | Medium ejected | * | [DH]=0, [DL]=(1:U-disk, 2:TF-card) | |
| 0x3C | 60 | Finished track on U-disk | * | [DH]=highByte(NUM), [DL]=lowByte(NUM) Not validated. Returns track number when song is finished on U-Disk | |
| 0x3D | 61 | Finished track on TF-card | * | [DH]=highByte(NUM), [DL]=lowByte(NUM) Returns track number when song is finished on TF | |
| 0x3E | 62 | Finished track on Flash | * | [DH]=highByte(NUM), [DL]=lowByte(NUM) Not validated. Returns track number when song is finished on Flash | |
| 0x3F | 63 | Initialization parameters | * | [DH]=0, [DL]= 0 - 0x0F. Returned code when Reset (0x12) is used. (each bit represent one device of the low-four bits) See Datasheet. 0x02 is TF-card. Error 0x01 when no medium is inserted. | |
| 0x40 | 64 | Error | ? | [DH]=0, [DL]= 0-7 Error code(Returned codes not yet analyzed) | |
| 0x41 | 65 | Reply | ? | [DH]=0, [DL]= 0-? Return code when command feedback is high 0x00 no Error (Other returned code not known) | |
| 0x42 | 66 | The current status | * | [DH] = Device number [DL] = 0 no play, 1 play | |
| 0x43 | 67 | The current volume | * | [DH]=0, [DL]= Volume (0x00-0x30) | |
| 0x44 | 68 | The current EQ | * | [DH]=0, [DL]= EQ(0/1/2/3/4/5) [Normal/Pop/Rock/Jazz/Classic/Base] | |
| 0x45 | 69 | The current playback mode | * | [DH]=0, [DL]= (0x00: no CMD 0x08 used, 0x02: CMD 0x08 used, not usefull) | |
| 0x46 | 70 | The current software version | * | [DH]=0, [DL]= Software version. (My version is 5) | |
| 0x47 | 71 | The total number of U-disk files | * | [DH]=highByte(NUM), [DL]=lowByte(NUM). Not validated | |
| 0x48 | 72 | The total number of TF-card files | * | [DH]=highByte(NUM), [DL]=lowByte(NUM) | |
| 0x49 | 73 | The total number of flash files | * | [DH]=highByte(NUM), [DL]=lowByte(NUM). Not validated | |
| 0x4A | 74 | Keep on (Datasheet) | ? | Unknown. No returned parameter | |
| 0x4B | 75 | The current track of U-Disk | * | [DH]=highByte(NUM), [DL]=lowByte(NUM), Current track on all media | |
| 0x4C | 76 | The current track of TF card | * | [DH]=highByte(NUM), [DL]=lowByte(NUM), Current track on all media | |
| 0x4D | 77 | The current track of Flash | * | [DH]=highByte(NUM), [DL]=lowByte(NUM), Current track on all media | |
| 0x4E | 78 | Folder "01" [DH]=x, [DL]=1 | * | [DH]=0, [DL]=(NUM) Change to first track in folder "01" Returns number of files in folder "01" | |
| 0x4F | 79 | The total number of folders | * | [DH]=0, [DL]=(NUM), Total number of folders, including root directory | |

This software is free to use and free to share

Additional info can be found on DFRobot site, but is not very reliable
Additional info:http://www.dfrobot.com/index.php?route=product/product&product_id=1121

Ype Brada 2015-04-06
*/

```
#include "SoftwareSerial.h"
# define Start_Byte 0x7E
# define Version_Byte 0xFF
# define Command_Length 0x06
```

```

# define End_Byte      0xEF
# define Acknowledge   0x00      //Returns info with command 0x41 [0x01: info, 0x00: no info]

SoftwareSerial mySerial(10, 11);

void setup () {
  Serial.begin(9600);
  mySerial.begin (9600);
  execute_CMD(0x3F, 0x00, 0x00); // Send request for initialization parameters
  while (mySerial.available()<10) // Wait until initialization parameters are received (10 bytes)
    delay(30); // Pretty long delays between successive commands needed

  // Set sound (0x06) to very low volume (0x05). Adept according used speaker and required volume
  execute_CMD(0x06, 0x00, 0x05);
}

void loop () {
  if (Serial.available())
  {
    // Input Serial monitor: Command and the two parameters in DECIMAL numbers (NOT HEX)
    // E.g. 3,0,1 (or 3 0 1 or 3;0;1) will play first track on the TF-card
    byte Command = Serial.parseInt();
    byte Parameter1 = Serial.parseInt();
    byte Parameter2 = Serial.parseInt();

    // Write your input at the screen
    Serial.print("Command : 0x");if (Command < 16) Serial.print("0"); Serial.print(Command, HEX);
    Serial.print(""); Serial.print(Command, DEC);
    Serial.print(""); Parameter: 0x";if (Parameter1 < 16) Serial.print("0");Serial.print(Parameter1, HEX);
    Serial.print(""); Serial.print(Parameter1, DEC);
    Serial.print("", 0x");if (Parameter2 < 16) Serial.print("0");Serial.print(Parameter2, HEX);
    Serial.print(""); Serial.print(Parameter2, DEC);Serial.println("");

    // Execute the entered command and parameters
    execute_CMD (Command, Parameter1, Parameter2);
  }

  if (mySerial.available()>=10)
  {
    // There is at least 1 returned message (10 bytes each)
    // Read the returned code
    byte Returned[10];
    for (byte k=0; k<10; k++)
      Returned[k] = mySerial.read();

    // Write the returned code to the screen
    Serial.print("Returned: 0x"); if (Returned[3] < 16) Serial.print("0"); Serial.print(Returned[3],HEX);
    Serial.print(""); Serial.print(Returned[3], DEC);
    Serial.print(""); Parameter: 0x"; if (Returned[5] < 16) Serial.print("0"); Serial.print(Returned[5],HEX);
    Serial.print(""); Serial.print(Returned[5], DEC);
    Serial.print("", 0x"); if (Returned[6] < 16) Serial.print("0"); Serial.print(Returned[6],HEX);
    Serial.print(""); Serial.print(Returned[6], DEC); Serial.println("");
  }
}

void execute_CMD(byte CMD, byte Par1, byte Par2)
// Execute the command and parameters
{
  // Calculate the checksum (2 bytes)
  word checksum = -(Version_Byte + Command_Length + CMD + Acknowledge + Par1 + Par2);
  // Build the command line
  byte Command_line[10] = { Start_Byte, Version_Byte, Command_Length, CMD, Acknowledge,
    Par1, Par2, highByte(checksum), lowByte(checksum), End_Byte};
  //Send the command line to the module
  for (byte k=0; k<10; k++)
  {
    mySerial.write( Command_line[k]);
  }
}

```

Préparer les fichiers

Les dossiers sont à nommer de 01 à 99, ils peuvent contenir des fichiers numérotés de 001 à 255, sous cette forme :

```

/01/001.mp3 # commande pour le jouer : myDFPlayer.playFolder(1, 1);
/01/002.mp3 # commande pour le jouer : myDFPlayer.playFolder(1, 2);
/01/003.mp3 # etc.

```

On peut préparer les fichiers avec ffmpeg :

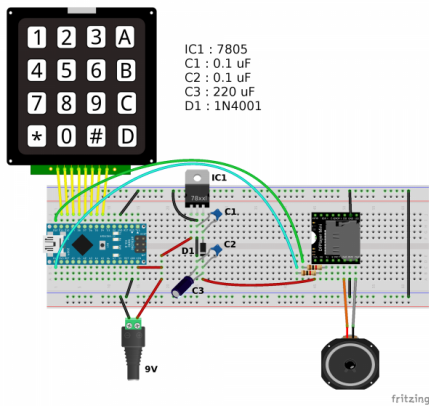
```

#!/bin/bash
for video in *.wav
do
  ffmpeg -i $video -vn -ar 44100 -ac 2 -ab 320k -f mp3 01/${video%.*}.mp3
done

```

Montage complet

Le montage complet est alimenté en 9V par un transfo externe, dont la tension est transformée en 5V pour alimenter l'arduino. Le fil d'alimentation est enroulé (deux spires) autour d'un anneau de ferrite pour réduire les parasites électromagnétiques qui parasitent le son. L'afficheur à 4 chiffres n'est pas utilisé et la sortie audio se fait sur le haut-parleur du combiné.



telephone_BMR_complet.ino (cliquer pour afficher le code)

[telephone_BMR_complet.ino](#)

```
/* Téléphone BMR version 6
 * SANS affichage 4 x 7 digits
 * décodage de clavier matriciel
 * et sortie audio
 * soit sur sortie jack TRRS, cf. ci-dessous myDFPlayer.enableDAC();
 * soit sur sortie combiné, avec l'ampli intégré au DFPlayer mini
 * version 6 : détecte quand même le clavier si pas de connection avec le DFPlayer
 * Quimper, La Baleine, 6 mars 2018, pierre@lesporteslogiques.net
 */

// *****
// Définitions pour l'afficheur 7 segments à 4 chiffres
// *****
#include "SevenSegmentTM1637.h"

const byte PIN_CLK = 2; // define CLK pin (any digital pin)
const byte PIN_DIO = 3; // define DIO pin (any digital pin)
SevenSegmentTM1637 display(PIN_CLK, PIN_DIO);
*/
// *****
// Définitions pour le clavier matrice
#include <Keypad.h>

const byte ROWS = 4; // quatre rangées
const byte COLS = 4; // quatre colonnes
char keys[ROWS][COLS] = {
  {'1','2','3','Z'},
  {'4','5','6','F'},
  {'7','8','9','Z'},
  {'*','0','#','R'}
};

/* Correspondances entre les broches identifiées sur le téléphone et les pins de l'arduino
R1 : 7      C1 : 8
R2 : 6      C2 : 9
R3 : 5      C3 : 10
R4 : 4      C4 : 11      */

byte rowPins[ROWS] = { 7, 6, 5, 4}; // connecter les rangées à ces pins
byte colPins[COLS] = { 8, 9, 10, 11}; // connecter les colonnes à ces pins

// *****
// Définitions pour le lecteur de son
#include "Arduino.h"
#include "SoftwareSerial.h"
#include "DFRobotDFPlayerMini.h"

SoftwareSerial mySoftwareSerial(12, 13); // RX, TX
DFRobotDFPlayerMini myDFPlayer;
void printDetail(uint8_t type, int value);

int DFPvolume = 18;

Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
```

```

void setup() {

  Serial.begin(9600);
  // Initialiser l'afficheur à 4 chiffres
  /*
  display.begin();           // initializes the display
  display.setBacklight(100); // set the brightness to 100 %
  display.print("INIT");     // display INIT on the display
  delay(1000);               // wait 1000 ms
  */

  mySoftwareSerial.begin(9600);
  Serial.println();
  Serial.println(F("DFRobot DFPlayer Mini Demo"));
  Serial.println(F("Initializing DFPlayer ... (May take 3-5 seconds)"));

  if (!myDFPlayer.begin(mySoftwareSerial)) { //Use softwareSerial to communicate with mp3.
    Serial.println(F("Unable to begin:"));
    Serial.println(F("1.Please recheck the connection!"));
    Serial.println(F("2.Please insert the SD card!"));
    //while(true);
    delay(1000);
    Serial.println(F("Lancement en mode test clavier"));
  } else {
    Serial.println(F("DFPlayer Mini online. "));

    myDFPlayer.setTimeout(500); //Set serial communication time out 500ms

    //myDFPlayer.enableDAC(); // pour branchement casque ou sortie ligne

    delay(500);

    myDFPlayer.volume(DFPvolume); //Set volume value. From 0 to 30

    delay(300);

    Serial.print("nb de fichiers presents dans le repertoire 01 : ");
    Serial.println(myDFPlayer.readFileCountsInFolder(1));
    //myDFPlayer.play(1); //Play the first mp3
  }
}

void loop() {

  static unsigned long timer = millis();

  char key = keypad.getKey();

  if (key){
    //display.print(key);
    Serial.println(key);
    if (millis() - timer > 1000) {
      playSound(key);
      timer = millis();
      //myDFPlayer.next(); //Play next mp3 every 3 second.
    }
    delay(300);

    if (myDFPlayer.available()) {
      printDetail(myDFPlayer.readType(), myDFPlayer.read()); //Print the detail message from DFPlayer to handle different errors
      and states.
    }
    /*else {
      display.clear();
    }*/
  }
}

void playSound(char key) {

  myDFPlayer.volume(DFPvolume);
  delay(100);

  switch(key) {
    case '0':
      myDFPlayer.playFolder(1, 1); //jouer le mp3 SD:/01/001.mp3; Folder Name(1-99); File Name(1-255)
      break;
    case '1':
      myDFPlayer.playFolder(1, 2);
      break;
    case '2':
      myDFPlayer.playFolder(1, 3);
      break;
    case '3':
      myDFPlayer.playFolder(1, 4);
      break;
    case '4':
      myDFPlayer.playFolder(1, 5);
      break;
    case '5':
      myDFPlayer.playFolder(1, 6);
      break;
    case '6':
      myDFPlayer.playFolder(1, 7);

```

```

        break;
    case '7':
        myDFPlayer.playFolder(1, 8);
        break;
    case '8':
        myDFPlayer.playFolder(1, 9);
        break;
    case '9':
        myDFPlayer.playFolder(1, 10);
        break;
    case '*':
        myDFPlayer.playFolder(1, 11);
        break;
    case '#':
        myDFPlayer.playFolder(1, 12);
        break;
    case 'F':
        myDFPlayer.playFolder(1, 13);
        break;
    case 'R':
        myDFPlayer.playFolder(1, 14);
        break;
    }
}

void printDetail(uint8_t type, int value){
    switch (type) {
        case Timeout:
            Serial.println(F("Time Out!"));
            break;
        case WrongStack:
            Serial.println(F("Stack Wrong!"));
            break;
        case DFPlayerCardInserted:
            Serial.println(F("Card Inserted!"));
            break;
        case DFPlayerCardRemoved:
            Serial.println(F("Card Removed!"));
            break;
        case DFPlayerCardOnline:
            Serial.println(F("Card Online!"));
            break;
        case DFPlayerPlayFinished:
            Serial.print(F("Number:"));
            Serial.print(value);
            Serial.println(F(" Play Finished!"));
            break;
        case DFPlayerError:
            Serial.print(F("DFPlayerError:"));
            switch (value) {
                case Busy:
                    Serial.println(F("Card not found"));
                    break;
                case Sleeping:
                    Serial.println(F("Sleeping"));
                    break;
                case SerialWrongStack:
                    Serial.println(F("Get Wrong Stack"));
                    break;
                case CheckSumNotMatch:
                    Serial.println(F("Check Sum Not Match"));
                    break;
                case FileIndexOut:
                    Serial.println(F("File Index Out of Bound"));
                    break;
                case FileMismatch:
                    Serial.println(F("Cannot Find File"));
                    break;
                case Advertise:
                    Serial.println(F("In Advertise"));
                    break;
                default:
                    break;
            }
            break;
        default:
            break;
    }
}
}

```

Sources et ressources

Bibliothèque arduino keypad : <http://playground.arduino.cc/Code/Keypad>

Bibliothèque pour afficheur 4 digits 7 segments de Bram Harmsen : <https://github.com/bremme/arduino-tm1637>

Utiliser un clavier matriciel : <https://playground.arduino.cc/Main/KeypadTutorial>

Fabricant de la puce pour lire les MP3 : <http://www.yxin18.com/kp/2015102450.html>

Fabricant du module DFPlayer mini : https://wiki.dfrobot.com/DFPlayer_Mini_SKU_DFR0299

Article extrait de : <http://www.lesporteslogiques.net/wiki/> - **WIKI Les Portes Logiques**
Adresse : http://www.lesporteslogiques.net/wiki/openatelier/projet/telephone_bmr?rev=1583239941
Article mis à jour: **2020/03/03 13:52**