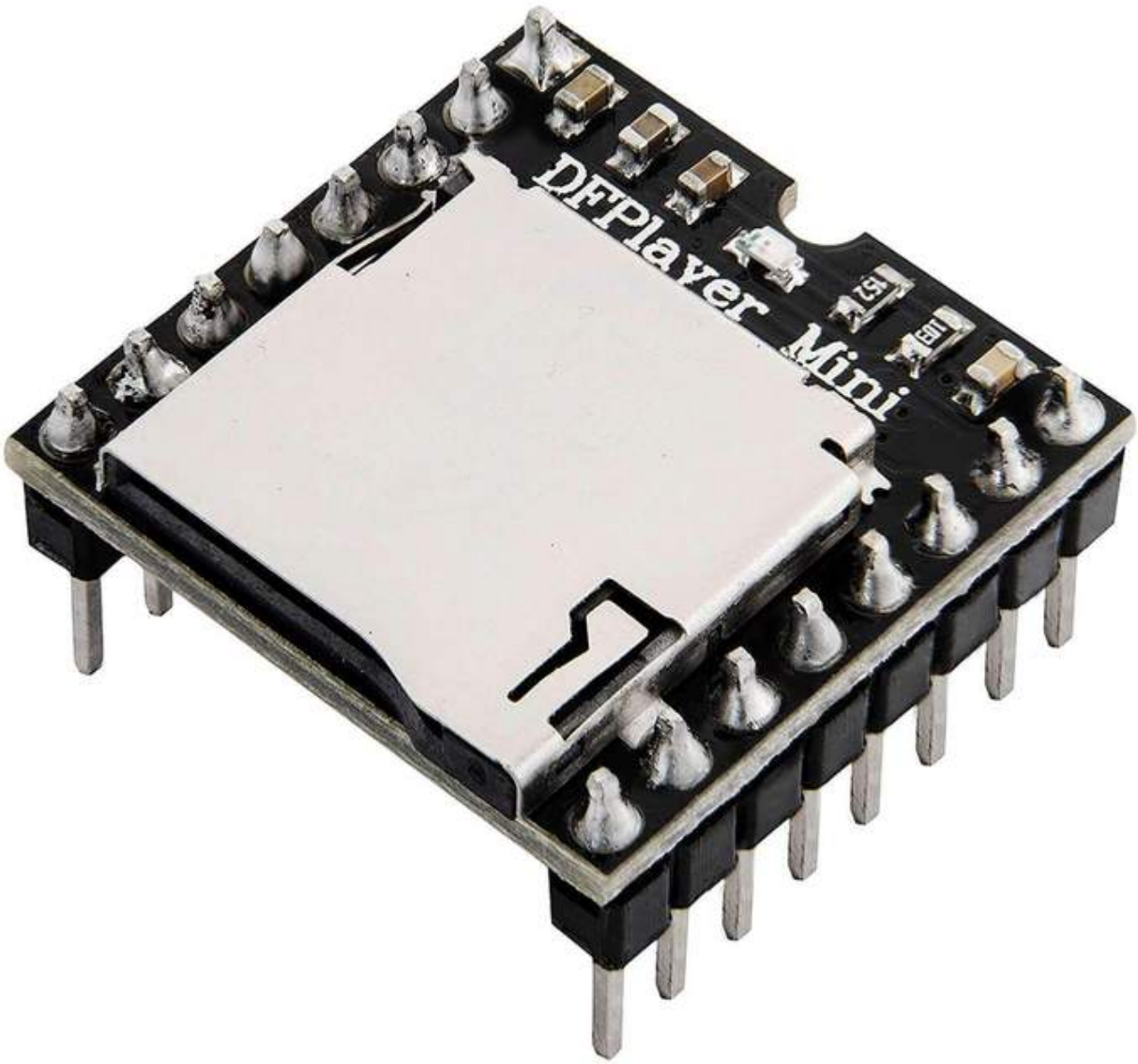
Willkommen!

Vielen Dank, dass sie sich für unser *MP3 DFPlayer Mini Modul* von *AZ-Delivery* entschieden haben. In den nachfolgenden Seiten werden wir Ihnen erklären wie Sie das Gerät einrichten und nutzen können.

Viel Spaß!



Das MP3-DFPlayer-Minimodul ist ein kleines und erschwingliches MP3-Modul, das direkt auf Lautsprecher oder Kopfhörer ausgeben kann. Das Modul kann als eigenständiges Modul mit angeschlossener Batterie, Lautsprecher und Drucktasten oder in Kombination mit jedem Arduino Board oder jedem anderen Board mit USART-Funktion verwendet werden.

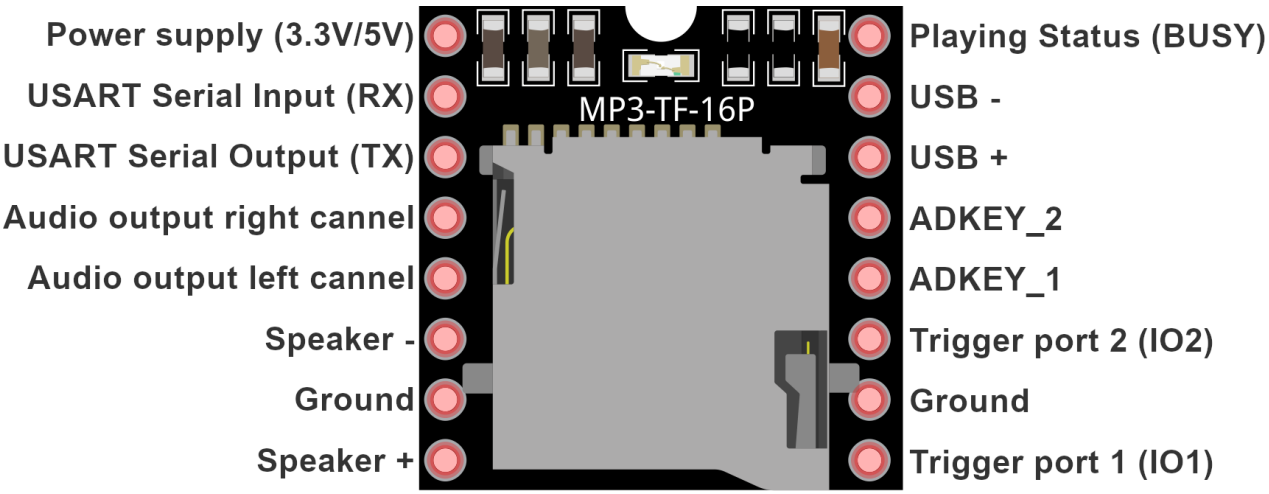
Das Modul unterstützt gängige Audioformate wie *MP3, WAV* und *WMA*. Außerdem unterstützt es TF-Karten mit *FAT16*- und *FAT32*-Dateisystem. Sie können Musik über eine einfache serielle Schnittstelle ohne komplexe Vorgänge abspielen.

Das Modul wird bereits mit zwei vorgelöteten *8*-poligen Stiftleisten geliefert und hat einen microSD-Kartensteckplatz auf dem Board. Auf dem Board befindet sich auch eine rote LED, die zur Anzeige des Abspielstatus von TXE Songs dient. Die LED ist mit dem *BUSY*-Pin des Moduls verbunden. Der *ON*-Zustand der LED zeigt an, dass das Lied gerade gespielt wird.

Technische Daten:

»	Betriebsspannungsbereich:	3.2V bis 5V DC
»	Standby-Stromt:	20mA
»	Betriebstemperatur:	-40℃ bis 70℃
»	UART-Anschluss:	Standard seriell (TTL level)
»	Baudrate:	Einstellbar (Standart 9.600)
»	Equalizer:	6 Stufen, einstellbar
»	Lautstärke:	30 Stufen, einstellbar
»	Abtastraten (kHz):	8/11.025/12/16/22/24/32/44.1/48
»	Ausgang:	- 24 Bit DAC Bereich 90dB, - SNR support 85dB
»	Ausgangsleistung:	3W
»	Widerstand d. Lautsprechers:	3Ω (Maximum 4Ω)
»	Dateisystem:	FAT16 oder FAT32
»	Maximaler Support:	- 32GB der TF Karte, - 32GB des USB-Flash-Laufwerks, - 64MB Bytes NORFLASH
»	Steuerungs-Modi:	- E/A Steuerungsmodus, - Serieller Modus, - AD-Taste Steuermodus
»	Wartefunktion für Werbetöne (die Musik kann angehalten werden; wenn die Werbung vorbei ist, wird die Musik weiter abgespielt)	
»	Audiodaten nach Ordnern sortiert. Das Modul unterstützt bis zu 100 Ordner und jeder Ordner kann bis zu 255 Lieder enthalten	

Pinbelegung



USART-Pins werden für die serielle Kommunikation verwendet. Wenn Sie ein hohes Rauschen feststellen, schließen Sie einen *1kΩ* Resisor seriell an den TX-Pin an.

Audio-Ausgangskanal-Pins werden als DAC-Pins (Digital-Analog-Wandler) verwendet, und Sie sollten sie an den externen Verstärker anschließen.

Lautsprecher-Pins sind Pins vom *3W*-Verstärker auf dem Board, und Sie können sie direkt an den externen Lautsprecher anschließen (8Ω max).

ADKEY-Pins werden für den AD-Steuerungsmodus verwendet.

Trigger-Port-Pins werden zum Einstellen des Lautstärkepegels oder zum Umschalten von Songs (Hardware) verwendet. Diese Pins sind aktiv *LOW*

USB-Pins werden für den Anschluss an den USB-Flash-Speicherstick verwendet.

Stromversorgung und BUSY-Pin

Das Modul unterstützt einen Betriebsspannungsbereich von *3,2V* bis zu *5V* DC. Schließen Sie die externe Stromversorgung zwischen Stromversorgungs-Pin und Ground-Pin an.

Die Spannung des *TTL*-Logikpegels am seriellen Port des Moduls beträgt *3,3V*. Wenn Sie also einen *5V*-Pegel verwenden (z.B. Uno Board), schließen Sie seriell einen Widerstand mit mehr als *1kΩ* Widerstand an den *RX*-Pin des Moduls an. Wenn Sie diesen Widerstand nicht verwenden, werden Sie hohes Rauschen auf den Audio-Ausgangskanälen (Lautsprecher) feststellen.

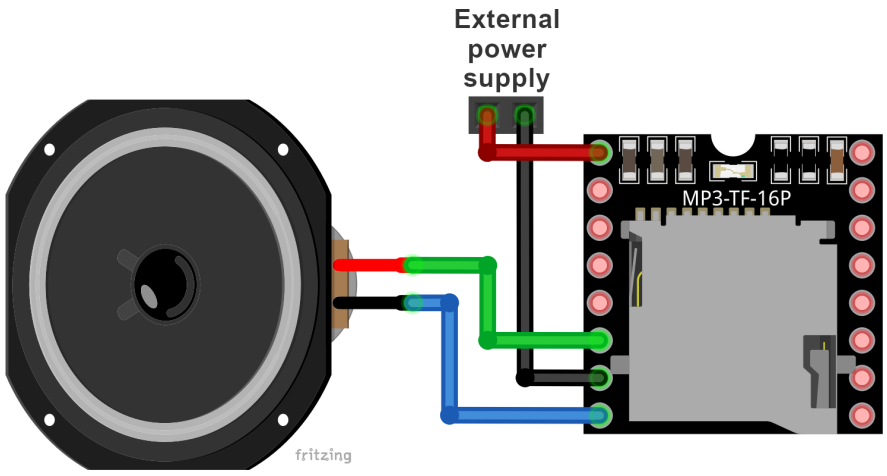
Der Wiedergabestatus oder *BUSY*-Pin wird verwendet, um den Wiedergabestatus eines Liedes anzuzeigen. Der *LOW*-Status an diesem Pin zeigt an, dass das Modul gerade einen Song abspielt, und der *HIGH*-Status zeigt an, dass das Modul keinen Song abspielt.

Audio-Ausgangskanäle und Lautsprecher-Pins

Der Hauptchip des Moduls verfügt über einen *24*-Bit-Digital-Analog-Wandler (kurz DAC). Der Chip hat zwei DAC-Pins, die direkt mit den Audio-Ausgangskanälen des Moduls verbunden sind. Sie sollten einen externen Verstärker an die Audio-Ausgangskanal-Pins anschließen, um die DAC-Fähigkeiten des Moduls zu nutzen. Wenn Sie diese Pins verwenden möchten, aktivieren Sie zunächst die DAC-Ausgabe, indem Sie einen entsprechenden Befehl an den Chip senden (s. später im Text).

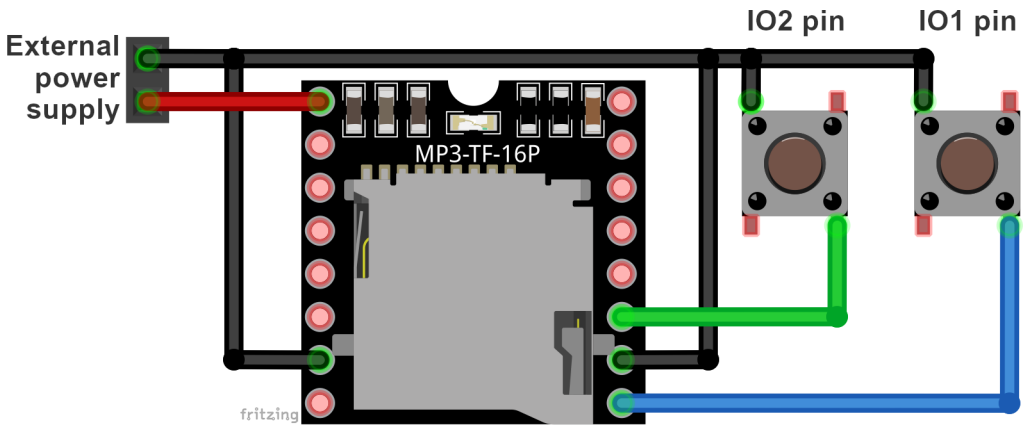
Auf dem Board des Moduls befindet sich eine *3W*-Verstärkerschaltung. Das Herz dieser Schaltung ist ein Gerät namens "*8002 audio amplifier*", eine *8*-polige integrierte Schaltung (IC), deren Ausgang mit den Pins *“Speaker+”* und “*Speaker-”* verbunden ist. Die Ausgabe des IC 8002 ist Monosound.

Schließen Sie den externen Lautsprecher an das Modul an, wie unten abgebildet:



Trigger-Ports (IO-Pins)

Diese Pins werden zum Titelwechsel und zum Einstellen der Lautstärkepegel über die Hardware verwendet. Schließen Sie Druckknöpfe an diese Pins an, wie unten abgebildet:



Wenn sich der Druckknopf im Ruhezustand befindet, sind die diagonalen Pins des Knopfes nicht angeschlossen. Wenn Sie den Druckknopf drücken, werden die diagonalen Pins des Druckknopfes verbunden, wodurch der Druckknopf in einen aktiven Zustand versetzt wird.

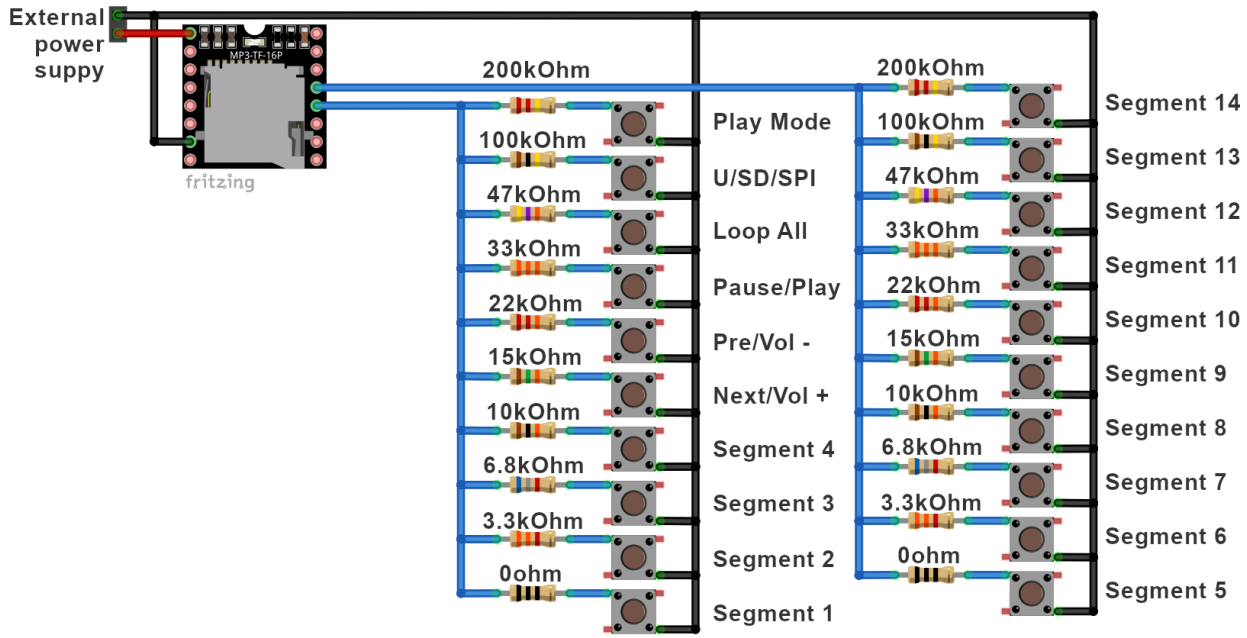
Ein kurzer Druck auf den *IO2*-Druckknopf spielt das nächste Lied. Ein langer Druck auf den *IO2*-Druckknopf erhöht die Lautstärke.

Ein kurzer Druck auf den *IO1*-Druckknopf spielt das vorherige Lied. Ein langer Druck auf die *IO2*-Drucktaste senkt den Lautstärkepegel.

Die Dauer des langen Drucks beträgt mehr als eine Sekunde. Alles, was kürzer als eine Sekunde ist, gilt als kurzes Drücken.

ADKEY-Pins

Die AD-Funktionalität des Chips auf dem Board des Moduls ermöglicht es Ihnen, 20 Widerstände mit Tasten an zwei AD-Ports des Moduls anzuschließen, wie unten abgebildet:



HINWEIS: Eine möglichst stabile Stromversorgung ist hierfür notwendig!

Ein kurzer Druck auf die *Play Mode*-Taste schaltet die Wiedergabe auf *interrupted* oder *not interrupted*. Das bedeutet, dass die Wiedergabe mit Werbung unterbrochen oder nicht unterbrochen wird. Eine lange Druckfunktion für diese Taste gibt es nicht.

Ein kurzer Druck auf die *U/TF/SPI*-Taste schaltet das Wiedergabegerät auf eine der folgenden Funktionen: *U* = USB-Flash-Disk, *TF* = SD-Karte, *SPI* = NORFLASH oder *Sleep*. Eine lange Druckfunktion für diese Taste gibt es nicht.

Ein kurzer Druck auf die *Loop All*-Taste schaltet den Wiedergabemodus um, um *alle Lieder zu loopen* oder *keins zu loopen*. Es gibt keine lange Druckfunktion für diese Taste.

Ein kurzer Druck auf die Taste *Pause/Play* hält den aktuell ausgewählten Song an oder spielt ihn ab. Für diese Taste gibt es keine Langdruckfunktion.

Ein kurzer Druck auf die Taste *Pre/Vol*+ spielt das vorherige Lied ab. Ein langer Druck auf die Taste Pre/Vol+ erhöht den Lautstärkepegel.

Ein kurzer Druck auf die Taste *Next/Vol-* spielt das nächste Lied ab. Ein langer Druck auf die Taste *Pre/Vol*+ verringert die Lautstärke.

Ein kurzer Druck auf die Taste *Segment1* spielt das Lied mit der Nummer 1. Durch langes Drücken der Taste *Segment1* wird das gleiche Lied wiederholt abgespielt.

Die Funktionen sind für alle anderen *Segment*-Tasten gleich, mit Ausnahme der Songnummer, die sich unterscheidet.

Serielle Schnittstelle

RX- und *TX*-Pins werden verwendet, um eine serielle Kommunikation mit einem externen Mikrocontroller herzustellen. Vergessen Sie nicht, einen Widerstand an den *RX*-Pin anzuschließen, wenn Sie die *5V-TTL*-Logik verwenden. Der serielle Anschluss des Moduls unterstützt den asynchronen seriellen Kommunikationsmodus. Die Standard-Baudrate der seriellen Kommunikation beträgt *9600 bps* und ist über die Software einstellbar.

Sie können die serielle Schnittstelle verwenden, um einfache Befehle an das Modul zu senden und somit viele Funktionen zu steuern, die das Modul unterstützt. Mehr über Befehle im nächsten Kapitel.

Technische Daten

Standart-Baudrate:	9600bps
Datenbits:	1
Checkout:	nein
Flow-Kontrolle:	nein

Format der Befehle

Um einen Befehl an das Modul zu senden, folgen Sie diesem Format:

$SB VB LB CMD ACK DATA1 DATA2 CHKS1 CHKS2 $EB

Mark	Byte	Byte Beschreibung
$SB	*0x7E*	Start-Byte
VB	*0xFF*	Versions-Byte
LB	*0xxx*	Die Anzahl der Bytes des Befehls ohne Start- und End-Bytes (In unserem Fall *0x06*)
CMD	*0xxx*	Sowie *PLAY* und *PAUSE* usw.
ACK	*0xxx*	Bestätigungs-Byte *0x00* = nicht ack, *0x01* = ack
DATA1	*0xxx*	Daten-High-Byte
DATA2	*0xxx*	Daten-Low-Byte
CHKS1	*0xxx*	Prüfsumme-High-Byte
CHKS2	*0xxx*	Prüfsumme-Low-Byte
$EB	*0xEF*	End-Byte

Das Bestätigungs-Byte wird verwendet, um Daten vom Modul zu erhalten. Wenn es auf 0x00 gesetzt ist, werden keine Daten vom Modul gesendet, und wenn es auf 0x01 gesetzt ist, erhalten Sie Antwortdaten vom Modul. Die Länge der Daten ist unbegrenzt, hat aber normalerweise zwei Bytes (*data1* und *data2* Bytes).

Um einen bestimmten Befehl zu senden, senden Sie einfach Byte für Byte seriell über die serielle Schnittstelle der Software (Sie können dies später im Code sehen).

Ordnerstruktur und Songnamen

Das Modul unterstützt verschiedene Arten von Ordnern und spezifische Namen für Songs. Die Namen der Ordner sind Nummern, mit Ausnahme von "*mp3*"- und "*ADVERT*"-Ordnern. Songnamen müssen mit einer Zahl beginnen, nach der die Zeichenfolge ohne Leerzeichen folgt. Beispiel für einen Songnamen:
`0001_Linking_Park_In_The_End.mp3`

Es gibt 5 Arten von Ordnern.

Der erste ist ein Ordnertyp, der *256* Songs enthalten kann. Das Modul unterstützt insgesamt *256* dieser Ordner. Die Namen dieser Ordner sind Zahlen im Bereich von *000* bis *255*. Die Songnamen in diesen Ordnern beginnen mit Nummern im Bereich von *000* bis *255*. Das Modul unterstützt keine Unterordner in diesen Ordnern.

Zweitens gibt es einen Ordnertyp, der *3000* Songs enthalten kann. Das Modul unterstützt insgesamt *16* dieser Ordner. Die Namen dieser Ordner sind Nummern im Bereich von *00* bis *15*. Die Songnamen in diesen Ordnern beginnen mit Nummern im Bereich von *0000* bis *2999*. Das Modul unterstützt keine Unterordner in diesen Ordnern.

Drittens gibt es einen Ordner namens "*mp3*", der ebenfalls *3000* Titel enthalten kann. Das Modul unterstützt insgesamt einen "*mp3*"-Ordner. Die Titelnamen in diesem Ordner beginnen mit Nummern im Bereich von *0000* bis *2999*. Das Modul unterstützt keine Unterordner in diesem Ordner.

Viertens gibt es einen Ordner namens "*ADVERT*", der ebenfalls *3000* Songs enthalten kann und für Werbesongs verwendet wird. Das Modul unterstützt insgesamt einen "*ADVERT*"-Ordner. Die Songnamen in diesem Ordner beginnen mit Nummern im Bereich von *0000* bis *2999*. Das Modul unterstützt keine Unterordner in diesem Ordner.

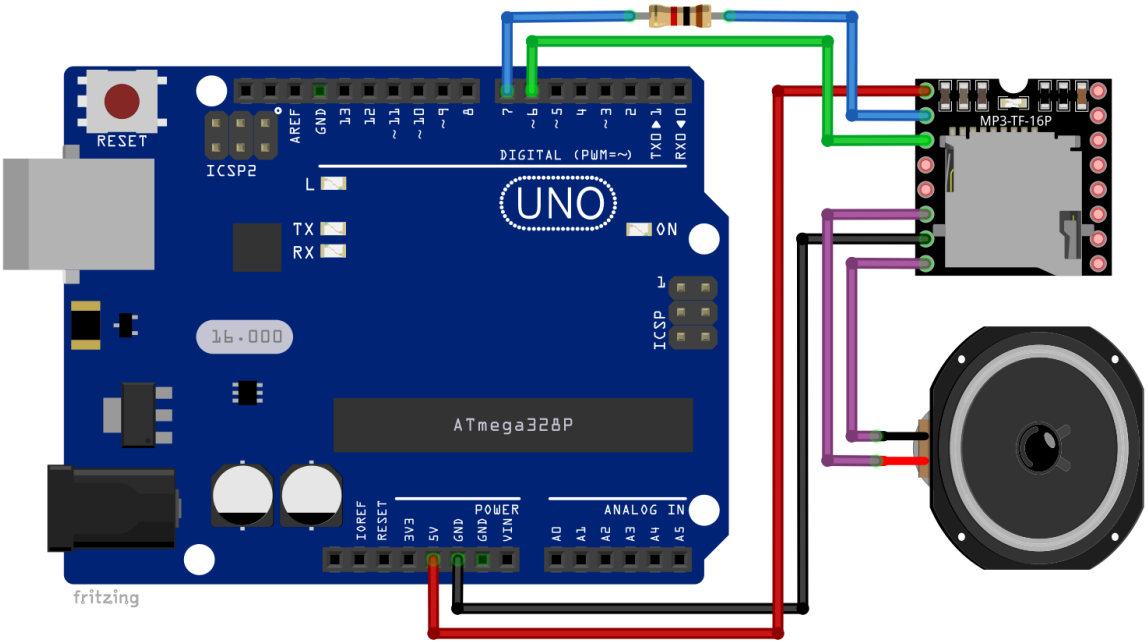
Und der fünfte Ordnertyp heißt "*root*". Wenn dies der einzige Ordner auf der SD-Karte oder dem USB-Speicher ist (keine anderen Ordner), kann dieser Ordner bis zu *65536* Songs enthalten. Dieser Ordner kann Unterordner haben, darunter fallen jegliche Ordnertypen. Der "*root*"-Ordner kann gleichzeitig Songs und Unterordner enthalten.

Beispiel für die Ordnerstruktur:

```
root
--- 0001r.mp3
--- 0002r.mp3
--- 0003r.mp3
--- 0004r.mp3
--- 0001
--- --- 0001x.mp3
--- --- 0002x.mp3
--- 0002
--- --- 0001y.mp3
--- mp3
--- --- 0001m.mp3
--- --- 0002m.mp3
--- --- 0003m.mp3
--- ADVERT
--- --- 0001a.mp3
--- --- 0002a.mp3
```

Befehle an das Modul senden

Um Befehle an das Modul zu senden, verbinden Sie das Modul mit dem Uno, wie unten abgebildet:



Modul Pin	**>**	**Uno Pin**	
VCC	>	5V	**Roter Draht**
RX	>	D7 (über 1kΩ Widerstand)	**Blauer Draht**
TX	>	D6	**Grüner Draht**
GND	>	GND	**Scharzer Draht**

Module Pin	**>**	**Lautsprecher Pin**	
SPK -	>	Eine Seite des Lautsprechers	**Lila Draht**
SPK+	>	And. Seite des Lautsprechers	**Lila Draht**

Hinweis: Sie können jedes andere Board mit einem Mikrocontroller verwenden, der USART-Fähigkeiten bestitzt.

Wir verwenden eine serielle Schnittstelle, die in der Software erstellt wurde, auf den digitalen I/O-Pins 6 und 7 des Uno, weil der Uno serielle Hardware-Pins (digitale I/O-Pins 0 und 1) für die Programmierung des Hauptmikrocontrollers verwendet.

Der Mikrocontroller kann keine Befehle zur Steuerung des Moduls senden, bis die Initialisierung des Moduls abgeschlossen ist und Daten zurückgesendet werden. Andernfalls werden die vom Mikrocontroller gesendeten Befehle ignoriert, und dies wirkt sich auf den Initialisierungsprozess aus.

Wenn nicht anders angegeben (durch Senden eines Befehls nach der Initialisierung), liest das Modul beim Einschalten zuerst die SD-Karte, und sollte keine SD-Karte verfügbar sein, schaltet es auf die USB-Flash-Disk um.

Sketch-Beispiel:

```
#include "SoftwareSerial.h"
#define Start_Byte        0x7E
#define Version_Byte      0xFF
#define Command_Length    0x06
#define End_Byte          0xEF
// Returns info with command 0x41 [0x01: info, 0x00: no info]
#define Acknowledge       0x01
SoftwareSerial mySerial(6, 7);  //  RX, TX
byte receive_buffer[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
char data;  // Used for received commands from Serial Monitor
byte volume = 0x00;       // Used to store current volume level
bool mute_state = false;  // Used to toggle mute state

// Excecute the command and parameters
void execute_CMD(byte Command, byte Data1, byte Data2) {
  // Calculate the checksum (2 bytes)
  word Checksum = -( Version_Byte + Command_Length + Command +
                                    Acknowledge + Data1 + Data2);
  // Build the command
  byte command_line[10] = { Start_Byte, Version_Byte,
                            Command_Length, Command, Acknowledge,
                            Data1, Data2, 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]);
  }
}
```

```
void reset_rec_buf() {
  for(uint8_t i = 0; i < 10; i++) {
    receive_buffer[i] = 0;
  }
}

bool receive() {
  reset_rec_buf();
  if(mySerial.available() < 10) {
    return false;
  }
  for(uint8_t i = 0; i < 10; i++) {
    short b = mySerial.read();
    if(b == -1) {
      return false;
    }
    receive_buffer[i] = b;
  }
  // When you reset the module in software,
  // received buffer elements are shifted.
  // To correct that we do the following:
  short b = receive_buffer[0];
  for(uint8_t i = 0; i < 10; i++) {
    if(i == 9) {
      receive_buffer[i] = b;
    }
    else {
      receive_buffer[i] = receive_buffer[i+1];
    }
  } // End correcting receive_buffer
  return true;
}
```

```
void print_received(bool print_it) {
  if(print_it) {
    if(receive()) {
      for(uint8_t i = 0; i < 10; i++) {
        Serial.print(receive_buffer[i], HEX); Serial.print("\t");
      }
      Serial.println();
    }
  }
  else { receive(); }
}

void module_init() {
  execute_CMD(0x0C, 0, 0); delay(1000); // Reset the module
  print_received(false);   delay(100);
  Serial.print("SDON\t");
  print_received(true);    delay(100);
  playFirst();
  setVolume(0x09);
}

void play_first() {
  Serial.print("PLYFST\t");
  execute_CMD(0x03, 0, 1); delay(100); // Play first song
  print_received(false);   delay(100);
  execute_CMD(0x45, 0, 0); delay(100); // Get playback status
  print_received(false);   delay(100);
  print_received(true);    delay(100);
}
```

```
void set_volume(uint8_t volume) {
  Serial.print("SETVOL\t");
  execute_CMD(0x06, 0, volume); delay(100); // Set volume level
  print_received(false);        delay(100);
  execute_CMD(0x43, 0, 0);      delay(100); // Get volume level
  print_received(false);        delay(100);
  print_received(true);         delay(100);
}
void play_next() {
  Serial.print("NEXT\t");
  execute_CMD(0x01, 0, 0); delay(100);
  print_received(false);   delay(100);
  execute_CMD(0x4C, 0, 0); delay(100); // Get current song played
  print_received(false);   delay(100);
  print_received(true);    delay(100);
}
void mute() {
  mute_state = !mute_state;
  if(mute_state) {
    execute_CMD(0x43, 0, 0); delay(100); // Return volume leve
    print_received(false);   delay(100);
    print_received(false);   delay(100);
    volume = receive_buffer[6];
    Serial.print("MUTE\t");
    execute_CMD(0x06, 0, 0x00); delay(100); // Set volume level
    print_received(false);      delay(100);
    execute_CMD(0x43, 0, 0);    delay(100); // Get volume level
    print_received(false);      delay(100);
    print_received(true);       delay(100);
  }
```

```
// one tab
  else {
    Serial.print("VOL\t");
    execute_CMD(0x06, 0, volume); delay(100); // Set previous vol
    print_received(false);        delay(100);
    execute_CMD(0x43, 0, 0);      delay(100); // Get volume level
    print_received(false);        delay(100);
    print_received(true);         delay(100);
  }
}

void random_play() {
  // Random plays all songs, loops all, repeats songs in playback
  execute_CMD(0x18, 0, 0);
  delay(100);
  Serial.print("RNDM\t");
  print_received(false);
  delay(100);
  execute_CMD(0x4C, 0, 0); // Get current song played
  delay(100);
  print_received(false);
  delay(100);
  print_received(true);
  delay(100);
}
```

```
void query_status() {
  execute_CMD(0x42, 0, 0); delay(100); // Get status of module
  print_received(false);   delay(100);
  Serial.print("STATUS\t");
  print_received(true);    delay(100);
  execute_CMD(0x43, 0, 0); delay(100); // Get volume level
  print_received(false);   delay(100);
  Serial.print("VOLUME\t");
  print_received(true);    delay(100);
  execute_CMD(0x44, 0, 0); delay(100); // Get EQ status
  print_received(false);   delay(100);
  Serial.print("EQ\t");
  print_received(true);    delay(100);
  execute_CMD(0x45, 0, 0); delay(100); // Get playback status
  print_received(false);   delay(100);
  Serial.print("PLYBCK\t");
  print_received(true);    delay(100);
  execute_CMD(0x46, 0, 0); delay(100); // Get software version
  print_received(false);   delay(100);
  Serial.print("SFVER\t");
  print_received(true);    delay(100);
  // Get total number of files on storage device
  execute_CMD(0x48, 0, 0); delay(100);
  print_received(false);   delay(100);
  Serial.print("FILES\t");
  print_received(true);    delay(100);
  execute_CMD(0x4C, 0, 0); delay(100); // Get current song played
  print_received(false);   delay(100);
  Serial.print("CRRTRK\t");
  print_received(true);    delay(100);
}
```

```
void setup() {
  Serial.begin(115200);
  mySerial.begin(9600);  delay(1000);

  Serial.println("\nInitialization");
  module_init();
}
void loop() {
  print_received(true);

  while(Serial.available() > 0) {
    data = Serial.read();
    // Serial.println(data, HEX); // For debugging
    if(data != "/n") {
      if(data == 'N') {
        Serial.println("\nPlay next song");
        play_next();
      }
      else if(data == 'B') {
        Serial.println("\nRandom play");
        random_play();
      }
      //  .....
      else if(data == 'D') {
        Serial.println("\nQuerry status of the module");
        query_status();
      }
    }
  }
  delay(100);
}
```

Die Skizze beginnt mit der Einbeziehung einer Library namens “*SoftwareSerial.h*”.

Dann definieren wir fünf Makros. Diese Makros stellen die Befehlsbytes dar, die für alle Befehle gleich sind. Das erste Byte heißt ”*Start_Byte*” mit dem Wert *0x7E*, der zweite Byte heißt “*Version_Byte*” mit dem Wert *0xFF*, der dritte Byte heißt “*Command_Length*” mit dem Wert *0x06*, der vierte Byte heißt “*End_Byte*” mit dem Wert *0xEF* und der fünfte Byte heißt “*Acknowledge*” mit dem Wert *0x01*.

Dann instantiieren wir ein serielles Software-Objekt namens “*mySerial*” mit der Codezeile: `SoftwareSerial mySerial(6, 7);`
wobei *6* für den digitalen I/0-Pin vom Uno steht, an den der *RX*-Pin des Moduls angeschlossen ist, und *7* für den digitalen I/0-Pin vom Uno, an den der *TX*-Pin des Moduls angeschlossen ist.

Dann erstellen wir ein Array namens “*receive_buffer*”, das zehn Elemente hat. Die Elemente des Array “*receive_buffer*” repräsentieren Bytes, die vom Modul gesendet und vom Uno empfangen werden.

Danach erstellen wir drei Variablen. Die erste wird "*data*" genannt und dient zum Speichern von Befehlen, wenn wir sie vom Serial Monitor senden. Die zweite Variable wird "*volume*" genannt und wird verwendet, um den aktuellen Lautstärkepegel zu speichern, wenn wir den Befehl "*Mute*" senden. Die dritte Variable heißt "*mute_state*" und wird verwendet, um den Stummzustand des Moduls umzuschalten.

Dann erstellen wir mehrere Funktionen. Die erste Funktion heißt "*execute_CMD()*", die drei Argumente akzeptiert und keinen Wert zurückgibt. Die Funktion "*execute_CMD()*" wird verwendet, um Befehle an das Modul zu senden. Das erste Argument ist das Befehlsbyte, das zweite ist *data1* Byte und das dritte ist *data2* Byte des Befehls. Zu Beginn der Funktion *execute_CMD()* berechnen wir mit dieser Zeile des Codes Prüfsummenbytes:

```
Wort-Prüfsumme = -(Version_Byte + Command_Length + CMD +
                   Acknowledge + Par1 + Par2);
```

Dannach erstellen wir ein Befehlsarray, namens "*command_line*". Dieses hat zehn Elemente, die zehn Bytes des Befehls darstellen: *Start_Byte*, *Version_Byte*, *Command_Length*, *Command*, *Acknowledge*, *Data1*, *Data2*, *highByte(Checksum)*, *lowByte(Checksum)*, und *End_Byte*. Am Ende der *execute_CMD()* Funktion, verwenden wir einen *for* loop , um zehn Bytes nacheinander per Software seriell an das Modul zu senden.

Die nächste Funktion heißt "*reset_rec_buf()*" und wird verwendet, um alle Werte der Elemente in *receive_buffer* auf Null zu setzen oder den Puffer zurückzusetzen. Die Funktion *reset_rec_buf()* akzeptiert keine Argumente und gibt keinen Wert zurück.

Danach erstellen wir eine Funktion namens "*receive()*". Sie wird verwendet, um Bytes vom Modul zu empfangen und sie im Array *receive_buffer* zu speichern. Die *receive()* Funktion akzeptiert keine Argumente und gibt einen boolschen Wert. Am Anfag von *receive()* rufen wir *reset_rec_buf()* auf, um das Array *receive_buffer* zurückzusetzen. Dann prüfen wir, ob es Daten auf Software-Serial gibt und ob diese Daten zehn Bytes umfassen.

Wenn die Daten zehn Bytes betragen, verwenden wir *for* loop, um alle zehn Bytes zu lesen. Nach dem Lesen eines Bytes prüfen wir, ob sein Wert gültig ist, indem wir prüfen, ob er sich von "*-1*" unterscheidet. Wenn er anders ist, speichern wir seinen Wert in *receive_buffer*. Wenn eine der Prüfungen nicht erfüllt wird, ist der zurückgegebene boolesche Wert "*false*"; ansonsten ist der Wert "*true*". Wenn wir das Modul in der Software zurücksetzen, werden die Elemente in *receive_buffer* versoben, so dass wir das korrigieren müssen. Das machen wir mit folgender Codezeile:

```
short b = receive_buffer[0];
for(uint8_t i = 0; i < 10; i++) {
  if(i == 9) {
    receive_buffer[i] = b;
  }
  else {
    receive_buffer[i] = receive_buffer[i+1];
  }
}
```

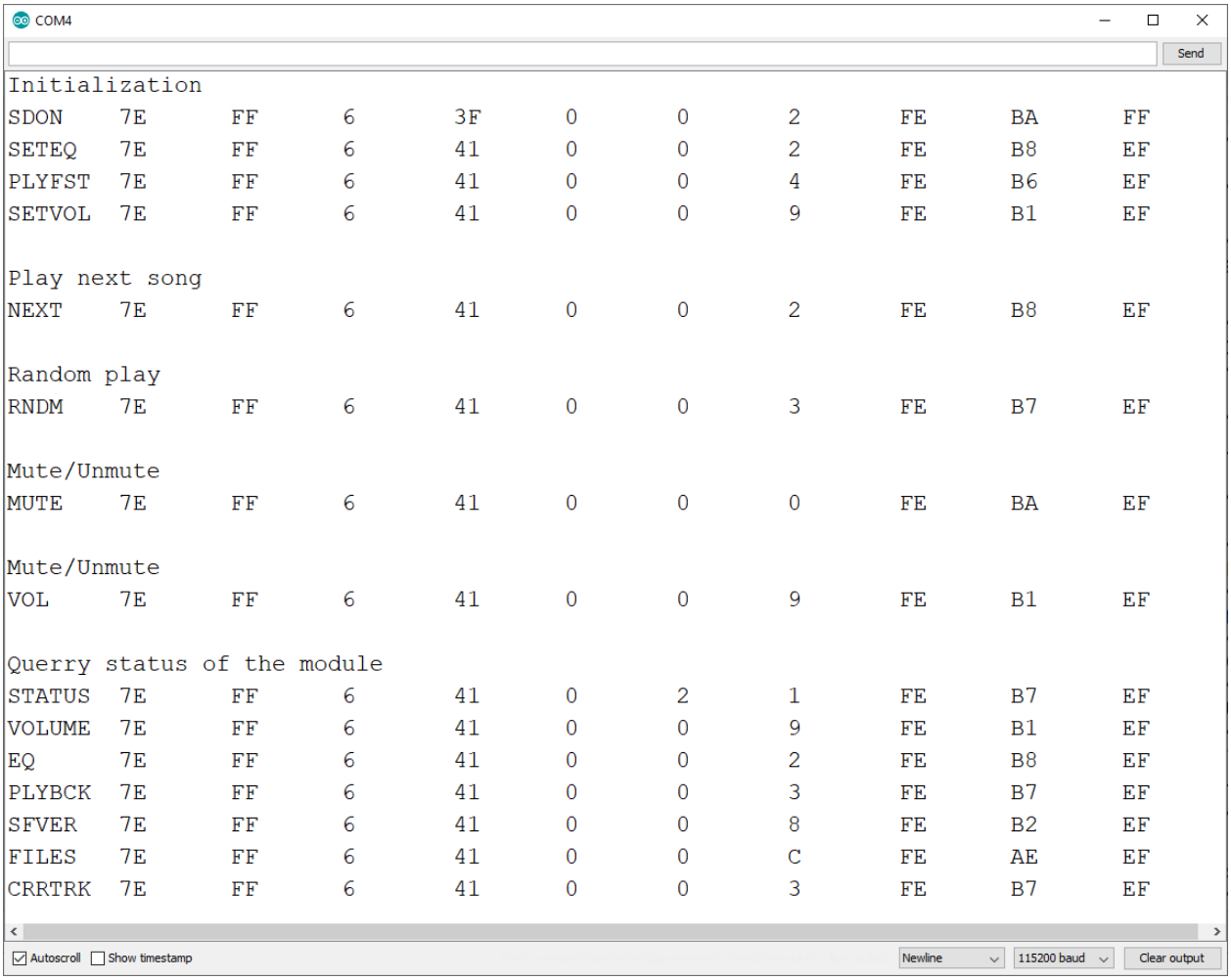
Danach erstellen wir die Funktion "*print_received()*" die zur Ausgabe der Daten auf dem Serial Monitor verwendet wird. Die Funktion *print_received()* akzeptiert als Argument einen booleschen Wert, der bei der Entscheidung, ob die Daten ausgegeben werden, verwendet wird. Wenn der Wert "*true*" ist, rufen wir die *receive()* Funktion und geben die Daten von *receive_buffer* aus. Wenn der Wert "*false*" ist, dann rufen wir die *receive()* Funktion auf, ohne die Daten auszugeben.
Nach diesen Funktionen erstellen wir zusätzliche Funktionen, die vorherige Funktionen beinhalten. Alle weiteren Funktionen sind selbsterklärend.

In der *setup()* Funktion starten wir die die Hardware seriell mit einer Baudrate von *115.200* bps, und Software seriell mit *9.600* bps (Standartbaudrate des Moduls). Dann rufen wir die Funktion *module_init()* auf, die das Modul initialisiert, Equalizer und Lautstärkepegel einstellt, den ersten Song auf dem Speichergerät abspielt und die Statusdaten auf dem seriellen Monitor ausgibt.

In der loop() Funktion warten wir auf die Daten auf der seriellen Hardware. Diese Daten werden vom Serial Monitor gesendet, wenn wir einen Befehl senden. Bei den Daten handelt es sich um einen der folgenden Buchstaben: N, B, D und mehrere andere Buchstaben. Wir prüfen, welcher Buchstabe gesendet wird, und rufen dann die entsprechende Funktion auf.

Der Sketch-Code in diesem eBook ist nur ein Beispiel. Ein Teil aus unserem Sketch-Beispiel. Wenn Sie den vollständigen Sketch sehen möchten, besuchen Sie das Repository unter dem folgenden *GitHub*-Link:
https://github.com/Slaveche90/DFPlayer_Custom_Sketch

Wenn Sie das komplette Sketch-Beispiel auf den Uno hochladen, starten Sie den Serial Monitor (*Tools* > *Serial Monitor*), und senden Sie einige Buchstaben aus dem Sketch über den Serial Monitor an den Uno. Die Ausgabe sollte wie unten abgebildet aussehen:



Befehls-Beispiele

Command	Bytes (HEX) *	Beschreibung
Next Song	7E FF 06 **01** 00 00 00 EF	Play next song
Previous Song	7E FF 06 **02** 00 00 00 EF	Play previous song
Play with index	7E FF 06 **03** 00 **00 01** EF	Play the first song
	7E FF 06 **03** 00 **00 02** EF	Play the second song
Volume up	7E FF 06 **04** 00 00 00 EF	Volume increase one level
Volume down	7E FF 06 **05** 00 00 00 EF	Volume decrease one level
Set volume	7E FF 06 **06** 00 **00 1E** EF	Set the volume to 30 (=0x1E)
Set EQ	7E FF 06 **07** 00 00 **02** EF	Set EQ to 02 – Rock; 00 / 01 / 02 / 03 / 04 / 05 Normal/Pop/Rock/Jazz/Classic/Base
Loop specific song	7E FF 06 **08** 00 **00 01** EF	Loop song 0001
Select device	7E FF 06 **09** 00 00 **01** EF	Select storage device to USB memeory
	7E FF 06 **09** 00 00 **02** EF	Select storage device to SD card
Sleep mode	7E FF 06 **0A** 00 00 00 EF	Chip enters sleep mode
Wake up	7E FF 06 **0B** 00 00 00 EF	Chip wakes up
Reset	7E FF 06 **0C** 00 00 00 EF	Chip reset
Play	7E FF 06 **0D** 00 00 00 EF	Resume the playback
Pause	7E FF 06 **0E** 00 00 00 EF	Playback is paused
Play specific song in a folder that supports 256 songs; module suports 256 folders (0 - 255) with 255 songs.	7E FF 06 **0F** 00 **01 01** EF	Play the song with the folder: 01/0001xxx.mp3
	7E FF 06 **0F** 00 **01 02** EF	Play the song in the folder: 01/0002xxx.mp3
Audio amplification	7E FF 06 **10** 00 **01 0A** EF	01 – Amp ON; 0A – level (0-31)
	7E FF 06 **10** 00 **00 00** EF	00 – Amp OFF
Loop all	7E FF 06 **11** 00 00 **01** EF	Start loop all songs
	7E FF 06 **11** 00 00 **00** EF	Stop looping all songs and stop playback
Play in mp3 folder	7E FF 06 **12** 00 **00 01** EF	Play song 0001 in mp3 folder (0x0001 – 0x0BB8; 3000 songs)
Play an add	7E FF 06 **13** 00 **00 01** EF	Play the song 0001 in folder ADVERT (0x0001 – 0x0BB8; 3000 songs)

*** Befehls-Bytes ohne zwei Prüfsummen-Bytes**

Command	Bytes (HEX) *	Beschreibung
Play specific song in a folder that supports 3000 songs; module suports 16 folders (0 - 15) with 3000 songs.	7E FF 06 **14** 00 **00 01** EF	In folder 0 play song 001
	7E FF 06 **14** 00 **91 11** EF	In folder 9 play song 273 (=0x111)
	7E FF 06 **14** 00 **F0 05** EF	In folder 15 (=0xF) play song 005
Stop playing add	7E FF 06 **15** 00 00 00 EF	Stop playing adverstment and resume previous playback
Enable loop all	7E FF 06 **16** 00 00 **01** EF	Enable loop all and start playing song 1
Stop play	7E FF 06 **16** 00 00 **00** EF	Stop the playback
Loop song in folder that supports 256 songs	7E FF 06 **17** 00 **01 02** EF	Loop song 02 in the 01 folder
Random playback	7E FF 06 **18** 00 00 00 EF	Random play all songs on the device
Set single loop play	7E FF 06 **19** 00 00 **00** EF	Start current song loop play
	7E FF 06 **19** 00 00 **01** EF	Stop current song loop play
Set DAC	7E FF 06 **1A** 00 00 **00** EF	Start DAC output
	7E FF 06 **1A** 00 00 **01** EF	Stop DAC output
Play specific song with volume	7E FF 06 **22** 00 **1E 01** EF	Set the volume to 30 (0x1E is 30) and play the first song
	7E FF 06 **22** 00 **0F 02** EF	Set the volume to 15 (0x0F is 15) and play the second song

*** Befehls-Bytes ohne zwei Prüfsummen-Bytes**

Status-Updates des Moduls

Es gibt eine Option, wenn Sie die Rückgabedaten aus dem Modul erhalten möchten. Diese Daten sind sehr nützlich, da sie Informationen über den aktuellen Wiedergabestatus, den Lautstärkepegel, die EQ-Option, den Zeitpunkt der Beendigung der aktuellen Wiedergabe des Songs usw. enthalten können. Um diese Option zu aktivieren, müssen Sie das *Acknowledge* Byte des Befehls auf den Wert *0x01* (*0x00* no return data) setzen.

Wenn Sie einen Befehl, bei dem Sie den *Acknowledge* Byte auf *0x01* gesetzt haben, kehren Daten zurück. Hier ist eine Liste der Befehle, die an Sie and das Modul senden können, um Status-Updates zu bekommen:

Command bytes (HEX) *	Beschreibung
7E FF 06 **3F** 00 00 00 EF	To get current storage device send this command
7E FF 06 **40** 00 00 **01** EF	This is return data, and it indicates error, where 01 is error value
7E FF 06 **41** 00 **00 00** EF	This is return data with no error. This indicates successifully received and executed command, where 00 00 is status of the module
7E FF 06 **42** 00 00 00 EF	To get playback status send this command
7E FF 06 **43** 00 00 00 EF	To get current volume level send this command
7E FF 06 **44** 00 00 00 EF	To get current EQ status send this command
7E FF 06 **47** 00 00 00 EF	To get total number of files on USB flash disk send this command
7E FF 06 **48** 00 00 00 EF	To get total number of files on SD card send this command
7E FF 06 **4B** 00 00 00 EF	To get current song number on USB flash disk send this command
7E FF 06 **4C** 00 00 00 EF	To get current song number on SD card send this command
7E FF 06 **4E** 00 00 00 EF	To get total number of files on any storage media send this command
7E FF 06 **4E** 00 00 **02** EF	To get total number of files in the folder 02 send this command
7E FF 06 **4E** 00 00 **0C** EF	To get total number of files in the folder 12 send this command
7E FF 06 **4F** 00 00 00 EF	To get total number of folders on any storage device send this command

*** Befehls-Bytes ohne zwei Prüfsummen-Bytes**

Return-Werte

Die zurückgegebenen Werte liegen in folgendem Format vor:

`0x7E 0xFF 0x06 **0x41** 0x00 **A** **B** checksum1 checksum0 0xEF`

Der Wert ***0x41*** zeigt an, dass ein Befehl vom Modul empfangen und erfolgreich ausgeführt wurde.

Der Wert "***A***" steht für Speichermedien, wobei:

A = *0x01* – USB-Flash-Laufwerk, und

A = *0x02* - SD-Karte.

Der Wert "***B***" zeigt den Status der Wiedergabe an, wobei

B = *0x00 anzeigt, dass die Wiedergabe gestoppt wird*

B = *0x01* anzeigt, dass die Wiedergabe läuft

B = *0x02* anzeigt, dass die Wiedergabe pausiert wird.

Beispiel für zurückgegebene Daten:

`0x7E 0xFF 0x06 **0x41** 0x00 **0x02** **0x01** 0xFE 0xF7 0xEF`

0x02 – Speichergerät ist eine SD-Karte

0x01 – die Wiedergabe läuft

Fehler

Wenn ein Fehler auftritt, sind die Daten in folgendem Format:
`0x7E 0xFF 0x06 **0x40** 0x00 0x00 **0x01** chks1 chks0 0xEF`

Wobei *0x40* anzeigt, dass ein Fehler aufgetreten ist, und `0x01` den Fehlerwert angibt.

Fehlerwerte mit Beschreibungen finden Sie in der folgenden Tabelle:

Error data (HEX) *	Beschreibung
7E FF 06 **40** 00 00 **01** EF	The module is busy
7E FF 06 **40** 00 00 **02** EF	The module is in sleep mode
7E FF 06 **40** 00 00 **03** EF	Serial reveiving error (frame is not received completely yet)
7E FF 06 **40** 00 00 **04** EF	Checksum incorret error
7E FF 06 **40** 00 00 **05** EF	Specified song is out of current songs scope
7E FF 06 **40** 00 00 **06** EF	Specified song is not found
7E FF 06 **40** 00 00 **07** EF	Intercut error (adverstment can only be played on playing song, not paused or stopped)
7E FF 06 **40** 00 00 **08** EF	SD card reading error (SD card is demaged or pulled out)
7E FF 06 **40** 00 00 **0A** EF	The module entered sleep mode

*** Befehls-Bytes ohne zwei Prüfsummen-Bytes**

Specific returned data

Wennd der Acknowledge Byte auf *0x01* gestellt wird, gibt das Modul Daten aus, wenn der Song beendet ist, wenn die SD-Karte (oder das USB-Flash-Laufwerk) *IN* oder *OUT* gedrückt oder herausgezogen wird oder wenn das Speichergerät online ist. Diese Werte werden zurückgegeben, ohne dass ein Befehl an das Modul gesendet wird.

Die Daten des Speichergeräts, wenn es auf *IN* gedrückt wird:
`0x7E 0xFF 0x06 **0x3A** 0x00 0x00 **A** 0xFE 0xF7 0xEF`
wobei:
0x3A anzeigt, dass das Speichergerät auf *IN* gedrück ist

Die Daten des Speichergeräts, wenn es *OUT* geschoben wird
`0x7E 0xFF 0x06 **0x3B** 0x00 0x00 **A** 0xFE 0xF7 0xEF`
wobei:
0x3B anzeigt, dass das Speichergerät *OUT* geschoben wird

A = *0x01* zeigt an, dass das Speichergerät eine USB-Flash-Disk ist
A = *0x02* zeigt an, dass das Speichergerät eine SD-Karte ist
A = *0x04* zegt an, ob das USB-Kabel mit dem PC verbunden ist oder nicht

Die Daten des fertigen Songs haben das folgende Format:
`0x7E 0xFF 0x06 **0x3D** 0x00 **0x00 0x05** 0xFE 0xF7 0xEF`
wobei: *0x3D* das Ende des Songs auf der SD-Karte anzeigt (*0x3C* = auf der USB-Flash-Disk),
0x00 0x05 den Songnamen "*0005*" anzeigt.

So sieht die Datenausgabe aus, wenn das Speichermedium online ist:

`0x7E 0xFF 0x06 **0x3F** 0x00 0x00  **A**  0xFE 0xF7 0xEF`

wobei:

0x3F den Onlinestatus anzeigt und

"***A***" kann verschiedene Werte haben

A = *0x01* die USB-Flash-Disk anzeigt

A = *0x02* die SD-Karte anzeigt

A = *0x03* die USB-Flash-Disk und die SD-Karte sind gleichzeitig online

A = *0x04* zeigt eine PC-Verbindung an

Der Rückgabewert der Wiedergabe

Wenn der Acknowledge Byte auf *0x01* gestellt wird, senden wir den Befehl für den Wiedergabestatus:

`0x7E 0xFF 0x06` **`0x45`** `0x00 0x00 0x00 chks1 chks0 0xEF`

Die Rückgabedaten sind in folgendem Format:

`0x7E 0xFF 0x06` **`0x41`** `0x00 0x00` **`A`** `chks1 chks0 0xEF`

wobei "***A***" mehrere verschiedene Werte haben kann:

A = *0x00* zeigt an, dass alle Songs auf dem Speichergerät nacheinander abgespielt und wiederholt werden,

A = *0x01* zeigt an, dass alle Songs in einem bestimmten Ordner nacheinander abgespielt und wiederholt werden,

A = *0x02* zeigt an, dass ein Song abgespielt und wiederholt wird,

A = *0x03* zeigt an, dass die Wiedergabe auf eine Zufallswiedergabe eingestellt ist und alle Songs auf dem Speichergerät wiederholt werden; bei einer Zufallswiedergabe werden die Songs wiederholt,

A = *0x04* zeigt an, dass ein Songs abgespielt wird, und wenn der Song beendet ist, stoppt die Wiedergabe.

Sie haben es geschafft. Sie können jetzt unser Modul für Ihre Projekte nutzen.

Jetzt sind Sie dran! Entwickeln Sie Ihre eigenen Projekte und Smart-Home Installationen. Wie Sie das bewerkstelligen können, zeigen wir Ihnen unkompliziert und verständlich auf unserem Blog. Dort bieten wir Ihnen Beispielskripte und Tutorials mit interessanten kleinen Projekten an, um schnell in die Welt der Mikroelektronik einzusteigen. Zusätzlich bietet Ihnen auch das Internet unzählige Möglichkeiten, um sich in Sachen Mikroelektronik weiterzubilden.

Falls Sie nach weiteren hochwertigen Produkten für Arduino und Raspberry Pi suchen, sind Sie bei AZ-Delivery Vertriebs GmbH goldrichtig. Wir bieten Ihnen zahlreiche Anwendungsbeispiele, ausführliche Installationsanleitungen, E-Books, Bibliotheken und natürlich die Unterstützung unserer technischen Experten.

https://az-delivery.de

Viel Spaß!

Impressum

https://az-delivery.de/pages/about-us