Utilisation via un µcontrôleur.

On peut commander le circuit par une liaison série avec un µcontrôleur...
… J'espère que les explications qui suivent vous sembleront claires et qu'il n'y a pas de coquilles…

La liaison série peut être établie de manière simple, dans un seul sens (de l'arduino vers le lecteur) et sans utiliser le "feedback" ou "retour" du circuit... Je l'utilise de cette manière… Même sans utiliser de retour par la liaison série, on pourrait néanmoins et assez facilement utiliser la pine "Busy" qui permet de vérifier si le lecteur est en "lecture" ou en "stand by". (Cela correspond à l'état de la led bleue du lecteur...)

La liaison peut aussi être une communication à double sens entre le MCU (Micro Controller Unit) et le lecteur. C'est de cette façon que les librairies ou "bibliothèques" disponibles fonctionnent.
Ces bibliothèques permettent de commander le ou les lecteurs à votre guise sans vous soucier de tout ce qui suit…
Comme inconvénient, ces librairies augmentent sensiblement la taille mémoire du programme téléversé…

Le "datasheet" donne les séquences à établir pour commander le lecteur.
La vitesse est de 9600 bauds. Le "0x" que l'on voit devant toutes les valeurs signifie que ces valeurs sont exprimées en Hexadécimal ou dit plus simplement, en "Hexa."

Toujours en reprenant le "data sheet", les commandes se résument à un envoi de données à la suite les unes des autres, du type ci-dessous:

$S	**Ver**	**Len**	**CMD**	**Feedback**	**Prm1**	**Prm2**	**Checksum**	**$O**
Start	Version	Longueur	Type de Cde	0 ou 1	Précisent la commande		Code de Vérif	

Si CMD = 0x06 = C'est une commande de volume
Si CMD = 0x03 = C'est la commande de lecture d'un N°fichier

Comme exemple: La Commande du volume a une valeur donnée
0x7E, 0xFF, 0x06, 0x06, 0x00 , 0x00, 0x18 , 0xFE, 0xDD , 0xEF

Dans l'ordre:
- Le Byte de Start (c'est toujours EF)…
- La version (c'est toujours FF)…
- La longueur de la séquence (Start, End, checksum non comptés, c'est toujours 6)…
- La commande - un Byte ou "Octet" Voir les possibilités sur le "datasheet"
- Le Feedback (c'est toujours 0 si utilisation sans feedback ou retour.
- Deux Bytes à la suite-précisent la commande si besoin - Ici 0 et 18 donc 24 sur 30 (**)
- Deux Bytes de Checksum – "Haut" et "Bas" - Voir explications ci-dessous
- Le Byte de fin (c'est toujours EF)

** Dans cet exemple, la commande "0x06" est une commande volume. Dans ce cas, les paramètres qui suivent donnent la valeur du volume dont la valeur est choisie entre 0 et 30.
0 valeur de l'octet haut, 18 valeur de l'octet bas – Cela fait 18, mais c'est en Hexa… Et, donc en fait, 24 (en décimal)

Pages suivantes…
Vous pouvez voir une façon de faire, sans utilisation de "Librairie" ou "Bibliothèque" spécifique.
Dans ce cas, avant le "setup", vous rajoutez toutes les commandes que vous avez prévu d'utiliser et que vous pourrez ainsi rappeler dans votre programme aux moments opportuns…

La liaison série n'y est utilisée que dans un seul sens, de l'Arduino (Tx) au lecteur mp3 (patte Rx)
Cette façon de faire simple utilise moins de place mémoire pour le programme

A la page 9, un exemple de syntaxe tirée d'un programme qui peut vous aider pour le vôtre…

Vous trouvez ci-dessous, des commandes que vous pouvez utiliser en tout ou en partie suivant votre besoin, dans le programme que vous écrirez. Ils se placent avant la partie "set up" du programme de l'Arduino, ce qui permet de les rappeler pour les utiliser dans la partie "Loop" du programme.
Le checksum a été déjà calculé pour chacune de ces lignes de commande
La page suivante donne le détail pour établir soi-même des lignes de commande autres.
Si deux niveaux de volume suffisent dans votre programme, comme "15" et "29", vous n'insérez que les deux lignes correspondantes dans le programme… (Mi Volume et Volume fort)

// Cde de quelques valeurs de volumes (de 0 pour le minimum à 30 au maximum)

```
byte Volume02[] = {0x7E, 0xFF, 0x06, 0x06, 0x00 , 0x00, 0x02 , 0xFE, 0xF3 , 0xEF } ;
byte Volume04[] = {0x7E, 0xFF, 0x06, 0x06, 0x00 , 0x00, 0x04 , 0xFE, 0xF1 , 0xEF } ;
byte Volume06[] = {0x7E, 0xFF, 0x06, 0x06, 0x00 , 0x00, 0x06 , 0xFE, 0xEF , 0xEF } ;
byte Volume08[] = {0x7E, 0xFF, 0x06, 0x06, 0x00 , 0x00, 0x08 , 0xFE, 0xED , 0xEF } ;
byte Volume10[] = {0x7E, 0xFF, 0x06, 0x06, 0x00 , 0x00, 0x0A , 0xFE, 0xEB , 0xEF } ;
byte Volume12[] = {0x7E, 0xFF, 0x06, 0x06, 0x00 , 0x00, 0x0C , 0xFE, 0xE9 , 0xEF } ;
byte Volume14[] = {0x7E, 0xFF, 0x06, 0x06, 0x00 , 0x00, 0x0E , 0xFE, 0xE7 , 0xEF } ;
byte Volume16[] = {0x7E, 0xFF, 0x06, 0x06, 0x00 , 0x00, 0x10 , 0xFE, 0xE5 , 0xEF } ;
byte Volume18[] = {0x7E, 0xFF, 0x06, 0x06, 0x00 , 0x00, 0x12 , 0xFE, 0xE3 , 0xEF } ;
byte Volume20[] = {0x7E, 0xFF, 0x06, 0x06, 0x00 , 0x00, 0x14 , 0xFE, 0xE1 , 0xEF } ;
byte Volume22[] = {0x7E, 0xFF, 0x06, 0x06, 0x00 , 0x00, 0x16 , 0xFE, 0xDF , 0xEF } ;
byte Volume24[] = {0x7E, 0xFF, 0x06, 0x06, 0x00 , 0x00, 0x18 , 0xFE, 0xDD , 0xEF } ;
byte Volume26[] = {0x7E, 0xFF, 0x06, 0x06, 0x00 , 0x00, 0x1A , 0xFE, 0xDB , 0xEF } ;
byte Volume29[] = {0x7E, 0xFF, 0x06, 0x06, 0x00 , 0x00, 0x1D , 0xFE, 0xD8 , 0xEF } ;
```

// Lecture directe de fichiers son numérotés de 01 à 24

```
byte Lecture01[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x01 , 0xFE, 0xF7 , 0xEF } ;
byte Lecture02[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x02 , 0xFE, 0xF6 , 0xEF } ;
byte Lecture03[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x03 , 0xFE, 0xF5 , 0xEF } ;
byte Lecture04[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x04 , 0xFE, 0xF4 , 0xEF } ;
byte Lecture05[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x05 , 0xFE, 0xF3 , 0xEF } ;
byte Lecture06[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x06 , 0xFE, 0xF2 , 0xEF } ;
byte Lecture07[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x07 , 0xFE, 0xF1 , 0xEF } ;
byte Lecture08[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x08 , 0xFE, 0xF0 , 0xEF } ;
byte Lecture09[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x09 , 0xFE, 0xEF , 0xEF } ;
byte Lecture10[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x0A , 0xFE, 0xEE , 0xEF } ;
byte Lecture11[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x0B , 0xFE, 0xED , 0xEF } ;
byte Lecture12[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x0C , 0xFE, 0xEC , 0xEF } ;
byte Lecture13[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x0D , 0xFE, 0xEB , 0xEF } ;
byte Lecture14[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x0E , 0xFE, 0xEA , 0xEF } ;
byte Lecture15[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x0F , 0xFE, 0xE9 , 0xEF } ;
byte Lecture16[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x10 , 0xFE, 0xE8 , 0xEF } ;
byte Lecture17[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x11 , 0xFE, 0xE7 , 0xEF } ;
byte Lecture18[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x12 , 0xFE, 0xE6 , 0xEF } ;
byte Lecture19[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x13 , 0xFE, 0xE5 , 0xEF } ;
byte Lecture20[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x14 , 0xFE, 0xE4 , 0xEF } ;
byte Lecture21[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x15 , 0xFE, 0xE3 , 0xEF } ;
byte Lecture22[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x16 , 0xFE, 0xE2 , 0xEF } ;
byte Lecture23[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x17 , 0xFE, 0xE1 , 0xEF } ;
byte Lecture24[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x18 , 0xFE, 0xE0 , 0xEF } ;
```

Autres commandes bien pratiques…
La commande "Reset" qui arrête une lecture en cours

0x7E, 0xFF, 0x06, 0x0C, 0x00 , 0x00, 0x00 , 0xFE, 0xEF , 0xEF

La commande "Repeat Play" …
(Les fichiers sont alors tous lus à la file et vont même reboucler à la toute fin de la lecture)

0x7E, 0xFF, 0x06, 0x11, 0x00 , 0x00, 0x01 , 0xFE, 0xE9 , 0xEF

A priori, quel que soit le N°de fichier lu auparav ant, la fonction "Repeat Play" fait démarrer du premier

Si vous ne connaissez pas et souhaitez mieux comprendre …. L'hexadécimal est une notation en base 16
En décimal (Base 10), on compte de 0 à 9. Ensuite, c'est 10, 11….
En Hexa, (base 16), on va d'abord compter de 0 à 9 puis on continue de A (10) à F (15)
Et donc 0x10 ou 10 en Hexa = 16 dans notre numérotation décimale 0x11= 17 0x20= 32….

En Décimal, on compte de 0 à 255 (Ce qui fait 256 valeurs différentes)
En Hexa, on compte de 0 à FF pour les mêmes valeurs

Comment se calcule le "Checksum" ou code de contrôle pour chaque ordre envoyé ? :

Dans la pratique, on peut utiliser la calculette **"Windows"** dans le mode **"Programmeur"**
(Accessible par le bouton affichage, on peut choisir standard, scientifique ou programmeur…)

Pour convertir sans erreur une valeur décimale en Hexa, exemple "22"
Cocher "Déc" et taper "22" puis cocher "Hex" et la valeur de 22 en Hexa apparaît "= 16"
La conversion est nécessaire pour commander un N°d e fichier, une valeur de volume ...

Le calcul du checksum se fait sur la partie verte de l'envoi...
0x7E, **0xFF, 0x06, 0x06, 0x00 , 0x00, 0x0C ,** 0xFE, 0xE9 , 0xEF
Il est calculé par la somme des bytes (le Start, le Checksum lui-même et le "End" sont exclus...)

Dans l'exemple ci-dessus, la commande est "**0x06".** Cela indique une commande de volume sonore
… sans demande de FeedBack ou "retour" de la part du circuit…
La valeur du volume est **0x00** (partie haute = 0) et **0x0C** (partie basse =12). Le Volume commandé est 12

On fait "0" moins cette somme et l'on garde les deux derniers Bytes
Dans l'exemple 0 - (FF+06+06+0+0+0C) = -------FEE9 donc FE (high) et E9 (Low)

Pour reprendre et mieux expliciter le calcul précédent du Checksum, il faut bien rester coché en "Hex"
Faire la somme de toutes les valeurs concernées
Dans l'exemple FF + 06 + 06 + 00 + 00 + 0C = 117 (Il est inutile de taper les zéros qui ne servent pas)
On tape ensuite 0 - 117 ce qui donne FFFFFFFFFFFFFEE9 - Il faut retenir les 4 derniers chiffres FEE9
C'est donc 0xFE puis 0xE9 qu'il faut écrire avant le 0xEF de fin

A noter... Les exemples du datasheet trouvé ne donnent pas un "checksum" correct…
Et, pour avoir essayé, il est clair que si le checksum est erroné, la cde n'est pas exécutée...

Page suivante, en exemple, des lignes tirées d'un programme qui exploite deux lecteurs
Cela peut vous permettre de mieux visualiser la syntaxe que vous devrez utiliser dans votre propre programme…

Un exemple avec les lignes importantes tirées d'un programme
- Sans librairie spécifique dédiée au lecteur -

```
#include <SoftwareSerial.h>

SoftwareSerial lecteur1 (11, 12);  // RX, TX  Lecteur Son destiné ambiance.
SoftwareSerial lecteur2 (9, 10);    // RX, TX  Lecteur Son destiné Sifflets, Sirène

……..
```

Utilisation de la Librairie "Software Serial"
Déclaration des noms "lecteur1" et "lecteur2"
Avec leurs broches associées

Ici, chacun des deux lecteurs utilisés est piloté par une liaison série crée via le soft de l'Arduino…

```
// Ci-dessous les lignes de commandes destinées aux lecteurs, mises avant le Setup, pour être reconnues ensuite
// Sous forme de tableau "Array", et sans préciser le nbre de caractères comptés ensuite par le soft (il y en a 10 pour chaque)

byte Lecture01[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x01 , 0xFE, 0xF7 , 0xEF } ;
byte Lecture02[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x02 , 0xFE, 0xF6 , 0xEF } ;
byte Lecture03[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x03 , 0xFE, 0xF5 , 0xEF } ;
byte Lecture04[] = {0x7E, 0xFF, 0x06, 0x03, 0x00 , 0x00, 0x04 , 0xFE, 0xF4 , 0xEF } ;
```

Déclaration de toutes les commandes qui seront utilisées au cours du programme
On peut donc en mettre moins, ou plus… C'est selon…

```
// Cde de volume
byte Volume08[] = {0x7E, 0xFF, 0x06, 0x06, 0x00 , 0x00, 0x08 , 0xFE, 0xED , 0xEF }  ;
byte Volume12[] = {0x7E, 0xFF, 0x06, 0x06, 0x00 , 0x00, 0x0C , 0xFE, 0xE9 , 0xEF } ;
byte Volume30[] = {0x7E, 0xFF, 0x06, 0x06, 0x00 , 0x00, 0x1E , 0xFE, 0xD7 , 0xEF }  ;


// Cde de répétition si besoin, à mettre après une cde de lecture
byte RepeatPlay[] = {0x7E, 0xFF, 0x06, 0x11, 0x00 , 0x00, 0x01 , 0xFE, 0xE9 , 0xEF } ;

// Reset
byte Reset[] = { 0x7E, 0xFF, 0x06, 0x0C, 0x00 , 0x00, 0x00 , 0xFE, 0xEF , 0xEF } ;
```

Ici, deux lecteurs sont utilisés

Dans cette façon de faire…
… Seules les broches 10 et 12 sont réellement utilisées. Chacune est reliée à la broche "Rx" du lecteur correspondant.

Par précaution, lors d'un câblage, on insère des résistances de 1K entre les broches 9 &10 et 11&12 du contrôleur et les broches Rx et Tx du lecteur.

```
void setup() {
…….
lecteur1.begin (9600) ;   // Liaison Série Soft lecteur 1   Pines 11 & 12
lecteur2.begin (9600) ;   // Liaison Série Soft lecteur 2   Pines 9 & 10

//  Arrêt des deux lecteurs par précaution au cas où ils seraient dans un mode de lecture
lecteur1.write (Reset, 10);
lecteur2.write (Reset, 10);
…….
}
```

Dans le Setup : démarrage communication
Vitesse de communication mise à 9600 Baud)
Envoi, par précaution, d'une commande initiale d'arrêt des 2 lecteurs…

La valeur 10 qui apparaît toujours dans la commande
Par ex (Reset, 10) ou (Volume30, 10)
Correspond au nombre d'octets envoyés – C'est toujours 10…

```
void loop() {
……
```

```
    // Mise en route et arrêt de la Musique d'ambiance via le Tx - Lecteur 1 utilisé
if ( RxSon1 > 1900 && SonAmb == 0 && STOP == 0 ) { SonAmb = 1 ; lecteur1.write (Volume30,10) ; lecteur1.write
(Lecture01,10); delay (100) ; lecteur1.write (RepeatPlay, 10) ; }
```

Ce sont des lignes extraites d'un programme ou en fonction du signal reçu, on envoie une commande soit au lecteur 1, soit au lecteur 2 (En vert et en gras)

```
……

// Une séquence d'arrêt
if ( RxSon1 > 1200 && RxSon1 < 1700 && ArrSon == 2 && SonAmb == 1 ) {  if ( (TempsSon+2500) < millis() ) { SonAmb = 0 ;
lecteur1.write (Reset, 10); delay (100) ; ArrSon = 0 ;} }

……

if ( millis() > ( TempsSiff + Retard1 ) && EnvoiCde == 0 && Seq1F == 1 ){ lecteur2.write (Volume30,10) ; lecteur2.write
(Lecture01,10); EnvoiCde = 1 ; }
                    ……
                                ……..}
```