

# Mémorisation de données sur une carte mémoire SD avec Arduino : Obtenir des informations sur les fichiers et les répertoires.



## Ateliers Arduino

par X. HINAULT

[www.mon-club-elec.fr](http://www.mon-club-elec.fr)



Tous droits réservés – 2012.

**Ce document légèrement payant est soumis au droit d'auteur et est réservé à l'usage personnel.**

Afin d'encourager la production de supports didactiques de qualité, ce document est légèrement payant.

La licence d'utilisation est attribuée pour un usage personnel uniquement, dans le cercle familial. Mise en ligne et diffusion non autorisées.

Si vous n'êtes pas le détenteur de la licence attribuée pour l'usage de ce document, soyez sympa, merci d'acheter votre exemplaire personnel ici : <https://monclubelec.dpdcart.com/>

Pour tout problème lié à l'utilisation de ce document, veuillez envoyer une copie ici : [support@mon-club-elec.fr](mailto:support@mon-club-elec.fr)

Pour obtenir tout autres types de licence d'utilisation (enseignement, commercial, etc...), veuillez contacter l'auteur ici : [support@mon-club-elec.fr](mailto:support@mon-club-elec.fr)

Vous avez constaté une erreur ? une coquille ? N'hésitez pas à nous le signaler à cette adresse : [support@mon-club-elec.fr](mailto:support@mon-club-elec.fr)

Truc d'utilisation : visualiser ce document en mode diaporama dans le visionneur PDF. Navigation avec les flèches HAUT / BAS ou la souris.

En mode fenêtre, activer le panneau latéral vous facilitera la navigation dans le document. Bonne lecture !

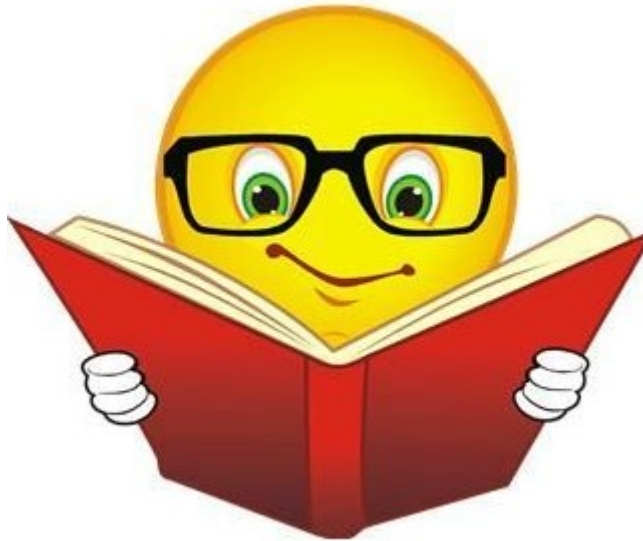
Lancer également le logiciel Arduino et connecter votre carte Arduino afin de pouvoir tester au fur et à mesure les codes d'exemples !

# 1. Intro

L'objectif ici est :

- d'apprendre à extraire une ligne ou un groupe de lignes voulus au sein d'un fichier de données sur une carte SD
- d'apprendre à obtenir des informations sur un fichier : taille, nombre de lignes,
- d'apprendre à afficher le contenu d'une carte mémoire SD avec Arduino
- de découvrir un usage avancé : obtenir des infos sur la carte SD.

... afin d'être en mesure de maîtriser la manipulation de données sur carte SD avec Arduino.



**Prêt ? C'est parti !**

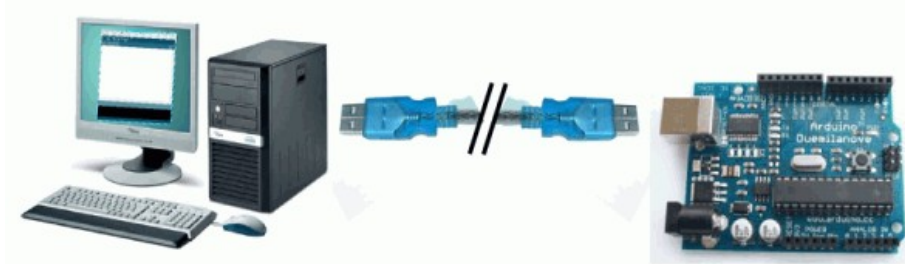
Une fois acquises les bases de l'utilisation d'une carte SD avec Arduino (voir tuto précédent), il va être intéressant de créer quelques fonctions plus élaborées pour obtenir de l'information sur un fichier, sélectionner des données précises, etc... C'est ce que nous allons faire ici.

**Suivez le guide !**

## 2. Matériel nécessaire pour les ateliers Arduino

Pour cet atelier, vous aurez besoin de tout ou partie des éléments suivants pour pouvoir réaliser les exemples proposés :

### De l'espace de développement Arduino

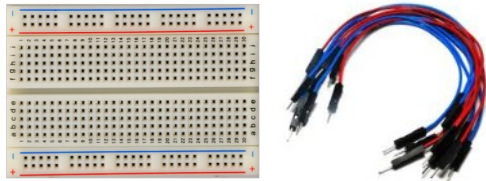


L'espace de développement Arduino associe :

- un ordinateur sous Windows, Mac Os X ou Gnu/Linux (Ubuntu)
- avec le logiciel Arduino installé (voir : <http://www.arduino.cc/>)
- un câble USB
- une carte Arduino UNO ou équivalente.

disponible chez : <http://shop.snootlab.com/> ou <http://www.gotronic.fr/>

### Du nécessaire pour réaliser des montages sans soudure

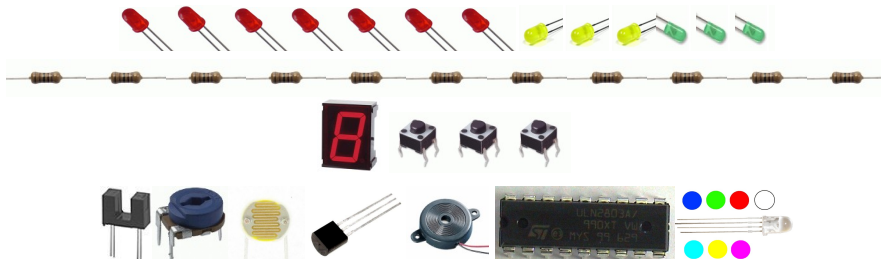


Pour réaliser des montages sans soudure, vous aurez besoin :

- d'une plaque d'essai ou breadboard moyenne (450 points)
- de quelques câbles souples (ou jumpers) mâle/mâle

disponible chez : <http://www.gotronic.fr/>

### De quelques composants de base



**Pour vous simplifier la vie, nous avons négocié ce kit pour vous !**

Vous pouvez commander ce kit complet directement en 1 clic chez notre partenaire

<http://www.gotronic.fr/> avec le code express **701710**

**GO TRONIC**  
ROBOTIQUE ET COMPOSANTS ÉLECTRONIQUES

Pour plus de détails, voir : [http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.ATELIERS](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ATELIERS)

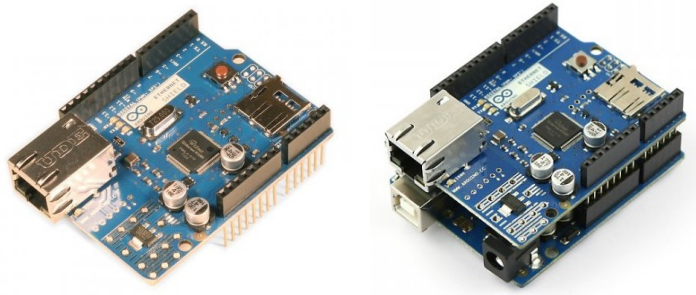
Pour les ateliers Arduino niveau débutant, vous devrez idéalement disposer des composants suivants :

- des LEDs 5mm Rouges(x20), Vertes (x5) et 3 Jaunes (x5)
- digit à cathode commune rouge 13mm (x1)
- Résistances (1/4w - 5%) de 270 Ohms (x20), 4,7K Ohms (x1), 1K Ohms (x1)
- mini bouton-poussoir (x3)
- Opto-fourche (x 1)
- Résistance variable linéaire 10K (x 1)
- Photo-résistance 7mm (x 1)
- Capteur de température LM35DZ (-55/+150°C - 10mV/°C) (x 1)
- Capsule son piézoélectrique (x 1)
- ULN 2803A (CI amplificateur 8 voies, 500mA/ voie) (x 1)
- LED 5mm multicolore RVB cathode commune (x 1)

### 3. Matériel spécifique nécessaire pour cet atelier

Pour cet atelier vous aurez besoin également shield disposant d'un étage carte mémoire SD. Plusieurs possibilités, notamment (non exhaustif) :

#### Soit d'une carte d'extension (shield) Ethernet (Arduino)



La carte d'extension (ou shield) ethernet Arduino est une carte électronique enfichable broche à broche sur la carte Arduino et qui permet d'utiliser Arduino sur un réseau ethernet local voire même sur internet.

Ce shield utilise la **communication SPI** (broches 13,12,11, et 10 +/- 4 ) pour communiquer avec Arduino.

**Ce shield intègre également un emplacement pour carte mémoire SD** pour des stockage de données ou de pages HTML locales.

Ne pas confondre ce shield avec la carte UNO Ethernet qui est une variante d'une carte UNO avec ethernet intégré.

disponible chez : <http://snootlab.com> ou <http://www.gotronic.fr/>

Prix constaté : 33€ environ

#### Soit d'une carte d'extension (shield) mémoire SD seule (Seeeduino)



La carte d'extension (ou shield) mémoire SD est une carte électronique enfichable broche à broche sur la carte Arduino et qui permet d'utiliser une carte mémoire micro SD, SD et SDHC.

Ce shield utilise la **communication SPI** (broches 13,12,11, et 10 ) pour communiquer avec Arduino.

disponible chez <http://www.gotronic.fr/> Prix constaté : 12€ environ

#### Soit d'une carte d'extension (shield) mémoire + temps réel (Snootlab)



La carte d'extension (ou shield) mémoire SD + « temps réel » est une carte électronique enfichable broche à broche sur la carte Arduino et qui dispose :

- d'un étage « carte mémoire SD » d'utiliser une carte mémoire micro SD, SD et SDHC. Cet étage utilise la **communication SPI** (broches 13,12,11, et 10 ) pour communiquer avec Arduino.
- d'un étage « temps-réel » basé sur un DS1307. Cet étage utilise la **communication I2C** (broches A4 et A5 ) pour communiquer avec Arduino.

disponible chez <http://snootlab.com/> Prix constaté : 18€ environ

**Noter que de nombreux autres shields disposent d'un étage carte mémoire SD, notamment les shields pour écrans TFT, etc...**

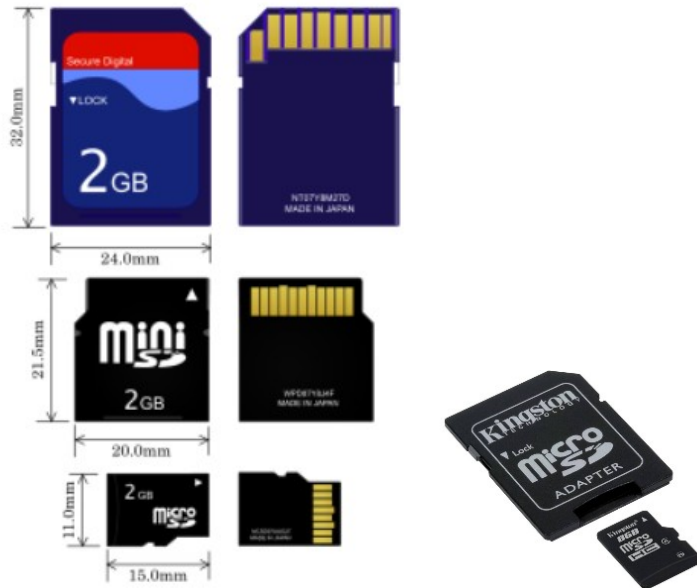
## 4. Pour info : les cartes mémoires SD

### C'est quoi une carte SD ?

- Je pense que vous le savez : il s'agit d'une carte mémoire permettant de stocker des données de façon non volatile.
- Ces cartes sont très répandues et sont utilisées notamment pour le stockage de photos avec les appareils photos numériques.

### Les différentes dimensions des cartes mémoire SD

- Les cartes mémoires SD existent dans plusieurs tailles différentes, du plus grand au plus petit :
  - le format SD
  - le format mini-SD
  - le format micro-SD
- Noter qu'il existe des adaptateurs micro-SD vers SD (pratiques !)



### Capacités des cartes mémoires SD

- Les SD-Card existent en toutes tailles : 2Go, 4Go, 8Go, 16Go, 32Go, 64Go...
- La véritable différence est le prix. Le bon compromis est un prix inférieur à 1€ / Go
- Avec Arduino, les cartes de plus de 4Go ne sont pas supportées.
- Les catégories de capacité des SD-Card sont indiquées par leur sigle :
  - SD : < 2Go
  - SDHC : 2 à 32 Go
  - SDXC : 32 à 2 To

### Vitesse d'échange d'une carte mémoire SD

- Un point plus méconnu est la vitesse d'échange de la carte SD. Ceci est indiqué par la "classe" de la SD Card, information indiquée sur l'emballage :
  - Les cartes SD classiques sont de classe 4 voire 6.
  - Pour avoir une vitesse d'échange accélérée, préférer une carte classe 10
- Voici les vitesses indicatives pour ces différentes classes, débit minimum garanti (et donc pouvant être supérieur) en écriture, plus élevé en lecture (x1,5 à x2) :
  - classe 4 : 4 Mo/sec
  - classe 6 : 6 Mo / sec
  - classe 10 : 10 Mo/sec

A titre de comparaison, les disques durs SSD actuels ont des vitesses en écriture de l'ordre de 250 Mo/sec et les disques durs classiques jusqu'à 100 Mo/sec.

L'utilisation d'une carte mémoire SD avec Arduino va permettre le stockage de grandes quantités de données (jusqu'à 4 Go à comparer aux 32Ko d'une carte UNO) et va permettre un stockage de données « non-volatiles », le tout sur un support directement utilisable sur un ordinateur classique au besoin.



## 5. Introduction à la manipulation de fichiers

### Introduction

- Le langage Arduino fournit la librairie SD qui va permettre la manipulation des répertoires et des fichiers au sein de la carte SD, comme sur n'importe quel disque dur ou clé USB, mais ici à partir d'Arduino !!
- Les manipulations effectuées ici sont de « bas niveau » et reprennent les principes de base de manipulation de fichier en vigueur sur la plupart des systèmes informatiques, notamment UNIX.

### Pour comprendre

- Il faut distinguer :
  - le **support matériel physique** qui stocke les données, la carte SD elle-même
  - la « partition » qui **est l'espace numérique** que vous avez créé en formatant la carte et dans lequel on va pouvoir créer des fichiers,
  - les **répertoires**, qui vont organiser et contenir les fichiers,
  - les **fichiers** qui sont des conteneurs de données,
  - les **données** elles-mêmes,
- Je vous propose une petite image, qui vaut ce qu'elle vaut, mais qui va vous aider à comprendre. Vous avez déjà mis le couvert je pense... :
  - la **table brute**, c'est votre carte SD, le support matériel
  - la **nappe**, c'est la « partition » qui vous permet de poser des choses sur votre table
  - les **assiettes**, ce sont les fichiers. Vous pouvez voir les répertoires comme de grandes assiettes ou plats dans lesquelles on pose les assiettes plus petites,
  - les données, c'est la **nourriture** que l'on met dans les assiettes...



### Tout est fichier !

- Par mesure de clarté, je distingue ici « répertoire » et « fichiers », mais en fait, en pratique, les répertoires sont également des fichiers ayant le rôle particulier de contenir d'autres fichiers, mais ce sont bel et bien des fichiers.
- Je pense que l'image des grandes assiettes dans lesquelles on met les plus petites illustre bien la chose...

### Nommage des fichiers au format « 8.3 »

- Un fichier, comme vous le savez probablement, a un nom de la forme « fichier.txt » :
  - sur votre ordinateur, rien ne vous empêche d'appeler votre fichier « mon\_fichier\_qui\_a\_un\_nom\_tres\_long.odt »
  - avec une carte mémoire SD utilisée avec Arduino, les noms de fichiers devront respecter le format dit « 8.3 » c'est à dire :
    - des noms en 8 lettres maxi,
    - un point et 3 lettres,
  - par exemple **filename.txt** est correct

### Principe d'accès à un fichier

- Quelque soit le système informatique, l'accès à un fichier suit à peu près la même procédure :
  - on commence par **ouvrir** le fichier (open )
  - on va pouvoir **écrire ou lire** des données dedans (write/read)
  - puis lorsque l'accès est terminé, on le **ferme** (close)
- Avec Arduino, le principe sera exactement le même : ouverture du fichier, opérations de lecture/écriture puis fermeture.

### Chemins des répertoires

- La localisation d'un fichier s'exprime sous la forme d'une chaîne, appelée « chemin » de la forme /ici/se/trouve/le/fichier.txt
- Chaque / correspond à un niveau de l'arborescence.
- Le **répertoire racine**, celui qui correspond à l'emplacement initial lorsque l'on ouvre la carte SD s'exprime sous la forme d'un seul /

**Vous comprendrez mieux cela par la suite si vous ne connaissez pas...**

## 6. Rappel : Langage Arduino : Introduction aux librairies

### C'est quoi une librairie Arduino ?

Le langage Arduino comporte de nombreuses instructions comme vous avez pu le constater, une quarantaine en tout. Ces instructions sont intégrées dans ce que l'on appelle le « noyau » ou « cœur » (core en Anglais) du langage Arduino. Ces instructions sont « générales » et servent souvent.

**Le langage Arduino peut cependant être étendu à la demande** avec des instructions dédiées à certaines applications particulières : afin de ne pas surcharger inutilement le « cœur », ces instructions spécifiques ont été intégrées dans des « paquets d'instructions » appelés librairies.

### Comment ça marche ?

Par exemple, si on utilise un afficheur LCD, un servomoteur ou encore si l'on utilise un shield ethernet (réseau), on va intégrer dans notre programme la librairie dédiée correspondante.

### Principe général d'utilisation

Pour intégrer une librairie dans un programme Arduino, c'est très simple : il suffit d'ajouter en début de programme une ligne de la forme :

```
#include <nomlibrairie.h> // librairie pour servomoteur
```

**ATTENTION : l'instruction include est un peu particulière : la ligne commence par un # et il n'y a pas de point virgule de fin de ligne !**

Ensuite, dans le code, au niveau de l'entête déclarative, là où vous déclarez vos variables, il va falloir déclarer un objet (une sorte de super variable) représentant la librairie. Cet objet est en fait une instance (= un exemplaire) d'une Classe (=le moule) qui regroupe les fonctions de la librairie. On a :

```
ClasseObjet monObjet; // déclare un objet
```

Généralement ensuite :

- au niveau de la fonction `setup()`, on initialise l'objet avec les paramètres voulus
- au niveau de la fonction `draw()`, on appelle les fonctions de la librairie sous la forme que vous connaissez déjà :

```
monobjet.fonction( param, param, ..);
```

**Rappel : pour utiliser une fonction d'une classe du langage Arduino, on utilise le nom de la classe + un point + le nom de la fonction.**

Il peut exister des variantes selon les librairies, mais grosso-modo, ça fonctionne de cette façon pour la plupart des librairies Arduino.

### Vous avez déjà utilisé une librairie !

Si vous êtes attentifs à tout ce qu'on a déjà vu, vous me direz que ça ressemble étrangement à l'utilisation de la classe **Serial...** et vous aurez raison ! En fait, la classe Serial est une librairie qui est intégrée implicitement lorsque vous lancez Arduino : c'est pour ça que vous n'avez pas besoin d'utiliser **#include** pour l'utiliser.

### Les librairies standards Arduino

Les librairies Arduino disponibles sont nombreuses, et disposent chacune de quelques fonctions à plusieurs dizaines... ce qui étend considérablement la puissance du langage Arduino et qui en fait aussi tout son intérêt. Voici la liste des librairies standards du langage Arduino (**le logiciel donne la liste...**) :

- [La librairie Serial](#) - pour les communications séries entre la carte Arduino et l'ordinateur ou d'autres composants
- [La librairie LCD](#) - pour l'utilisation et le contrôle d'un afficheur LCD alphanumérique standard.
- [La librairie Servo](#) - pour contrôler les servomoteurs.
- [La librairie Stepper](#) - pour contrôler les moteurs pas à pas (nécessite une interface de commande)
- [La librairie Ethernet](#) - pour se connecter à Internet en utilisant le module Arduino Ethernet
- [La librairie EEPROM](#) - référence - pour lire et écrire dans la mémoire EEPROM non volatile.
- [La librairie SD](#) - référence - pour utiliser une carte mémoire SD (utiliser des fichiers, stocker des données, ...)
- [La librairie SoftwareSerial \(Série Logicielle\)](#) - référence - pour communication série logicielle sur n'importe quelles broches de la carte Arduino
- [La librairie Wire / I2C](#) - référence - Interface "deux fils" (TWI/I2C) pour envoyer et recevoir des données sur un réseau de modules ou capteurs.
- [La librairie SPI \(Serial Peripheral Interface\)](#) - pour communication série avec des modules externes supportant le protocole SPI
- [Firmata](#) - pour communiquer avec des applications sur l'ordinateur utilisant un protocole série standard.

**En jaune les plus utiles.** Impressionnant non ? On les étudiera pas à pas...

### Les librairies de la communauté

A côté de ces librairies standards, il existe toute une série de librairies proposées par les uns et les autres et qui concernent des matériels spécifiques, ou autre. Par exemple :

- [La librairie Keypad](#) - pour l'utilisation des claviers matriciels. (hors référence).

Faites un tour ici pour voir ce qui existe : <http://arduino.cc/playground/Main/LibraryList>

## 7. La librairie Arduino SD

### Présentation

- La librairie Arduino SD permet de lire et d'écrire les cartes mémoires SD, à l'aide d'un shield dédié comportant un étage « carte SD », par exemple le shield « Ethernet »
- La librairie supporte les systèmes de fichier FAT16 et FAT 32 sur les cartes mémoires SD standard et les cartes SDHC.
- La librairie supporte l'ouverture d'un seul fichier à la fois et n'utilise que les noms de fichiers courts, dit « 8.3 »
- La communication entre la carte Arduino et la carte mémoire SD utilise la communication SPI, laquelle utilise les broches 11, 12 et 13 de la plupart des cartes Arduino.

Prenez quand même le temps de réaliser de quoi on parle : avec une simple carte Arduino, il va être possible d'utiliser un espace mémoire de plusieurs Go pour y stocker des données ! D'autre part, le support utilisé est polyvalent et pourra être lu directement sur un ordinateur classique.

### Inclusion

La librairie s'intègre dans un programme avec la ligne (pas de ; !!) :

```
#include <SD.h> // inclut la librairie pour carte mémoire SD
```

Noter que cette librairie nécessite également l'utilisation de la librairie de communication série SPI que l'on inclura avec la ligne :

```
#include <SPI.h> // inclut la librairie SPI
```

### Les classes de la librairie

A la différence de plusieurs autres librairie qui n'intègrent qu'une seule classe (une classe est un ensemble de fonctions regroupées), la librairie SD intègre deux classes :

- la classe **SD** qui fournit les fonctions pour accéder à la carte SD et manipuler ses fichiers et répertoires. L'objet SD est directement disponible à l'inclusion de la librairie.
- la classe **File** qui permet de lire et d'écrire dans les fichiers individuels sur la carte SD.



### Les fonctions de la librairie

Chaque classe dispose de plusieurs fonctions associées :

#### Classe SD (accès à la carte SD, manipulation des fichiers et répertoires)

- begin()** : initialise la carte SD avec la broche voulue – Renvoie True/False
- exists()** : teste si un répertoire ou un fichier existe sur la carte SD
- mkdir()** : crée un répertoire
- open()** : ouvre ou crée un fichier en écriture
- remove()** : efface un fichier
- rmdir()** : efface un répertoire

#### Classe File (lecture/écriture dans des fichiers individuels)

- available()** : teste si octets disponibles en lecture
- close()** : ferme le fichier
- flush()** : écrit physiquement les données dans le fichier
- peek()** : lit un octet dans le fichier sans passer à l'octet suivant
- position()** : renvoie position courante au sein du fichier
- print()** : écrit les données dans le fichier sans saut de ligne
- println()** : écrit les données dans le fichier avec saut de ligne
- seek()** : se place à la position voulue
- size()** : renvoie la taille du fichier en nombre d'octets
- read()** : lit un octet dans un fichier
- write()** : écrit un octet dans un fichier
- isDirectory()** : test si un fichier est un répertoire
- openNextFile()** : ouvre le fichier suivant
- rewindDirectory()** : se place au niveau du premier fichier/répertoire

### Exemple du principe d'utilisation

La structure type d'un programme utilisant une carte SD va être la suivante :

#### Au niveau de l'entête déclarative

- Inclusion des librairies **SPI** et **SD**
- Déclaration d'un objet **File**

#### Au niveau de la fonction **setup()**

- Initialisation de la librairie SD avec l'instruction **SD.begin(broche)**.

#### Au niveau de la fonction **loop()**

- A ce niveau on utilisera selon les besoins toutes les fonctions utiles de la librairie pour lire, écrire, manipuler les fichiers et répertoires.



## 8. Utiliser une carte mémoire SD avec Arduino : Préparatifs

### Acquérir une carte mémoire SD adaptée à votre shield

- La première chose à faire est d'acheter une carte mémoire SD si vous n'en n'avez pas une sous la main.
- Le point clé est d'avoir une carte adaptée à votre shield Arduino : certains shields disposent d'un connecteur pour carte micro-SD, d'autre pour une carte SD au format standard. De plus, le lecteur de carte sur votre poste fixe est souvent au format SD standard.
- **Le plus simple et le plus polyvalent à mon sens est donc d'utiliser une carte micro-SD + adaptateur SD.** Côté prix, compter quelques euros dans n'importe quelle grande surface ou « Electro-dépôt » local.



- Autre point important : la capacité de votre carte SD ! Nul besoin de choisir une carte de grande capacité : une carte de 1 ou 2Go fera largement l'affaire (et ça sera moins cher en plus!) . Pour éviter les problèmes de lecture avec la librairie Arduino, je vous conseille de ne pas aller au-delà de 4Go.

#### Remarque

A titre indicatif, avec Arduino, on ne va stocker que quelques octets à la fois. Admettons que nous stockions 20 octets pour un ensemble de 6 mesures analogiques et que l'on fasse un enregistrement toutes les secondes.... Sachant que 1Go, c'est 1 073 741 824 octets, avec une carte de **2Go**, on pourra stocker 107374182 groupes de 20 octets soit pendant une **durée de 3 ans et 5 mois !!** Et si on ne fait un enregistrement que **toutes les minutes, pendant une durée de.... 200 ans !**

Tout ça pour dire que vous avez de la marge !!

- Le choix de la classe n'a pas d'importance avec Arduino.

### Disposer d'un lecteur de carte SD sur son poste fixe

- C'est presque une lapalissade, mais je préfère le dire : vous avez besoin d'un poste fixe avec un lecteur de carte SD pour préparer votre carte, lire les fichiers sur votre poste fixe, etc...
- La plupart des ordinateurs récents possèdent un lecteur de carte SD en façade, ou une simple fente latérale sur un netbook (eeePC notamment)

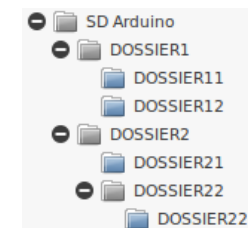


- Il existe également des lecteurs de carte SD externes à connecter sur port USB



### Formater et préparer la carte mémoire SD

- La première chose à faire est de formater la carte SD à l'aide de l'utilitaire de votre choix. Sous Gnu/Linux, vous pourrez utiliser l'utilitaire de disque.
- Le point clé : **formater la carte en format FAT, 16 ou 32.** Ne pas utiliser les formats NTFS, EXT, etc...
- Ensuite, créer quelques dossiers et fichiers texte vides :



**ATTENTION : Le formatage d'une carte SD efface toutes les données !**

## 9. Utiliser une carte mémoire SD avec Arduino : le montage

- Le montage est très simple et se résume à enficher le shield utilisé comportant un étage carte mémoire SD sur la carte Arduino, broche à broche. On pourra utiliser indifféremment l'un ou l'autre shield (=carte d'extension) disposant d'un étage carte mémoire SD :



- L'information importante à connaître est : **quelle broche de sélection est utilisée par l'étage mémoire du shield utilisé ?**
  - la broche 10 par défaut (sur le shield « Mémoire » de Snootlab par exemple)
  - la broche 4 sur le shield Ethernet (la 10 est utilisée par l'étage Ethernet)



Exemple avec le shield « Mémoire » de [Snootlab](#) qui intègre également un étage « temps réel » (RTC)

## 10. Extraire une ligne ou un groupe de lignes voulus au sein d'un fichier de données sur une carte SD

### Ce qu'on va faire ici...

- Imaginez à présent que vous avez enregistré des dizaines, voire des centaines de lignes de données issues de mesures dans votre fichier : **il va devenir assez laborieux et même impossible de stocker tout ceci dans un String pour manipuler les données** : la RAM de la carte Arduino n'est pas suffisante et votre programme va planter ! Grosso modo, pour quelques centaines de caractères, ça passera, au-delà, non...
- D'où l'idée de ne **manipuler les données dans un String** que par « petits bouts », autrement dit par ligne.
- Il est dès lors nécessaire de pouvoir **accéder au contenu d'une ligne du fichier très simplement à partir de son numéro**. Ceci n'est pas implémenté en natif dans la librairie SD, mais qu'à cela ne tienne, je vous propose ici une fonction clé en main de mon crû pour le faire !
- Pour l'essentiel, ce code reprend la même chose que les codes vus dans un tuto précédent, la différence réside dans l'ajout de la **fonction `getLine()` qui permet d'extraire une ligne sous forme d'un String**. Cette fonction reçoit en paramètre le nom du fichier et le numéro de ligne.
- Ce code est disponible ici : <https://gist.github.com/sensor56/b336167abcbe5cbf2480>

### Entête déclarative

#### Inclusion des librairies utiles

- On commence par inclure les librairies
  - la librairie Arduino **SD** qui permet d'utiliser une carte mémoire SD avec une carte Arduino
  - la librairie Arduino **SPI** qui permet d'utiliser la communication série SPI, ici utilisée par la carte SD. Noter qu'il n'est pas indispensable d'inclure cette librairie à ce niveau, ceci étant fait automatiquement par la librairie SD, mais par mesure de clarté, j'ai pris l'habitude de le faire.

#### Déclaration broche utilisée

- On déclare la broche utilisée pour sélectionner la carte SD pour la communication SPI (si vous ne savez pas ce qu'est la communication SPI, voir le tuto dédié) : cette broche est variable selon les shields avec étage mémoire SD utilisés (voir la doc de votre shield au besoin) :
  - soit la broche 10 par défaut (cas du shield Mémoire de Snootlab par exemple)
  - soit la broche 4 pour le shield Ethernet

#### Déclaration des objets utiles

- **Noter que l'on ne déclare pas l'objet SD**, qui est déclaré implicitement dans la librairie SD.

```
#include <SPI.h> // fichier librairie pour communication SPI utilisée par carte SD
#include <SD.h> // fichier librairie pour gestion de la carte SD

// broche de sélection de la carte SD : à adapter au shield utilisé
const int selectSD=10; // par défaut
//const int selectSD=4; // pour le shield Arduino
```

## Fonction **setup()**

### Configuration des broches numériques

- **Point important : même si cette broche n'est pas utilisée pour sélectionner la carte mémoire SD, il est impératif de mettre en sortie la broche /SS de la carte Arduino** (=broche 10 sur la UNO), sinon la communication SPI ne se fera pas correctement. C'est ce que nous faisons ici.

### Initialisation série

- On initialise la communication série à 115200 bauds

### Initialisation de la carte SD

- on affiche ensuite un message indiquant que l'initialisation de la carte SD est en cours
- puis on appelle la fonction **SD.begin()** en passant en paramètre la broche utilisée : noter que cette fonction est placée au sein d'une condition d'une boucle **if()** : ceci est possible car la fonction **SD.begin()** renvoie True si l'initialisation se passe bien et False dans le cas contraire.
  - si l'initialisation ne se réalise pas correctement (le ! inverse la condition) : alors on affiche le message d'échec,
  - puis on sort de la fonction **setup()** à l'aide de l'instruction **return**
- sinon, on se retrouve après la condition et on affiche un message de succès.

```
//--- fonction setup exécutée une fois au lancement
void setup(){

    // configuration des broches E/S
    pinMode(10, OUTPUT); // IMPORTANT ++ : laisser la broche /SS(=10) en sortie - obligatoire avec librairie SD

    // configuration de la communication série
    Serial.begin(115200); // utiliser le meme debit coté Terminal Serie

    //--- initialisation de la carte SD
    Serial.println("Initialisation de la carte SD en cours...");

    if (!SD.begin(selectSD)) { // si initialisation avec broche selectSD en tant que CS n'est pas réussie
        Serial.println("Echec initialisation!"); // message port Série
        return; // sort de setup()
    } // if SD begin()

    //--- si initialisation réussie : on se place ici :
    Serial.println("Initialisation reussie !"); // message port Série
```

## Fonction **setup()** (suite)

### Test existence fichier et suppression si il existe déjà

- A présent, nous commençons par déclarer un tableau de `char` correspondant au nom du fichier. **IL FAUT VRAIMENT FAIRE ATTENTION ET BIEN VERIFIER à ce que le format du nom du fichier soit bien en « 8.3 »** c'est à dire 8 lettres maxi pour le nom et 3 lettres maxi après le « . » L'écriture sur carte SD avec la librairie Arduino SD fonctionne très bien... sous réserve de bien respecter cette règle. Sinon, vous obtiendrez des messages d'erreur ou bien les opérations sur les fichiers ne se feront pas correctement !
- Ensuite, on teste si le fichier existe avec la fonction `SD.exists()` en passant en paramètre le nom du fichier. Cette fonction renvoie `True/False`, c'est pourquoi il est possible de la passer en condition à un `if()` de manière à exécuter le code voulu uniquement si le fichier existe :
  - ici si le fichier existe, nous l'effaçons à l'aide de la fonction `SD.remove()`
  - des messages série permettent de suivre les opérations...
  - ce qui donne :

```
//---- test existence fichier et suppression du fichier si il existe ---
char nomFichier[]="testFile.txt"; // utiliser un nom de fichier format 8.3 - doit etre un tableau de char

if (SD.exists(nomFichier)) { // si le fichier existe

    Serial.println("-----");
    Serial.print("Le fichier existe : "); // affiche message
    SD.remove(nomFichier); // efface le fichier
    Serial.println("Suppression du fichier."); // affiche message

} // fin si fichier existe
```



## Fonction **setup()** (suite)

### Création du fichier

- On commence par créer un objet File représentant un fichier ou un répertoire, à l'aide de la fonction **SD.open()**, ici en mode **FILE\_WRITE** : ouvre le fichier pour lecture et écriture, en démarrant au début du fichier.

### Écriture de données dans le fichier

- Une fois l'objet **File** créé, il est possible de tester son existence directement à l'aide d'une condition **if()** : un objet **File** non vide sera un équivalent de **True**. C'est ce que l'on commence par faire.
- Si le fichier existe (c'est à dire si il a correctement été ouvert) :
  - on ouvre une boucle **for** de comptage de 0 à 99 (pour 100 passages) et à chaque passage :
    - on commence par enregistrer le numéro de ligne (ou de mesure... un **index numérique** quoi...) suivi d'un ;
    - on enregistre ensuite la valeur de **millis()**, fonction qui renvoie le nombre de millisecondes écoulées depuis le lancement du programme, suivi d'un ;
    - on enregistre ensuite le résultat de la mesure analogique sur la broche A0 renvoyée par la fonction **analogRead()**, suivi d'un **saut de ligne**,
    - ... le tout à l'aide des fonctions **println()** et **print()** tout simplement !!
  - puis, une fois terminé, on ferme le fichier avec l'instruction **close()** de l'objet **File**. Remarquer au passage que la fonction **open()** est une fonction de l'objet **SD** et que la fonction **close()** est une fonction de l'objet **File**.

**Pour chaque mesure, on obtiendra une ligne de la forme « index ; time ; valeur »**

```
// création nouveau fichier
File dataFile=SD.open(nomFichier, FILE_WRITE); // crée / ouvre un objet fichier et l'ouvre en mode écriture - NOM FICHIER en 8.3 +++++
// un seul fichier ne peut etre ouvert à la fois - fermer au préalable tout fichier déjà ouvert

Serial.println("-----");

if (dataFile){ // le fichier est True si créé

    for (int i=0; i<100; i++) {

        dataFile.print(i+1); // la valeur du numéro de ligne
        dataFile.print(";"); // ; de séparation
        dataFile.print(millis()); // la valeur de millis()
        dataFile.print(";"); // ; de séparation
        dataFile.println(analogRead(A3)); // mesure sur la voie A0 + saut de ligne
        delay(10); // pause en ms entre chaque mesure...

    } // fin for

    dataFile.close(); // fermeture du fichier obligatoire après accès

} // si fichier existe
else { // sinon = si probleme creation
    Serial.println("Probleme creation fichier");
} // fin else
```

## Fonction **setup()** (suite)

### **Lecture d'une ligne dans le fichier**

- La lecture de n'importe quelle ligne du fichier devient très simple avec la fonction `getLine()` décrite ci-après : dans un `String`, on stocke le résultat de la fonction à laquelle on a passé en paramètre le numéro de ligne et le nom du fichier.
- Il suffit ensuite d'afficher la ligne.
- ce qui donne :

```
//----- obtenir une ligne voulue -----  
  
String ligne=getLine(nomFichier,92, true); // fonction lecture de ligne avec messages de debug  
  
Serial.println ("Ligne 92 : ");  
Serial.println(ligne);
```

### **Lecture d'un intervalle de lignes dans le fichier**

- De la même façon, il devient très simple d'afficher un ensemble de lignes voulues à l'aide d'une boucle `for` au sein de laquelle on appelle la fonction `getLine()`, ce qui donne :

```
//----- obtenir un intervalle de lignes voulues -----  
  
Serial.println ("Intervalle des lignes 22 -> 32 ");  
  
int debut=22;  
int fin=32;  
  
for (int j=debut; j<=fin; j++) {  
    ligne=getLine(nomFichier,j, false);  
    Serial.print(ligne);  
}  
  
} // fin setup
```

## Fonction **loop()**

- Laissée vide ici

```
//--- fonction loop exécutée en boucle infinie
void loop() {
    // laissée vide ici
} // fin loop
```

## Fonction **getLine()**

- Je vous laisse regarder le détail de la fonction largement commentée : la fonction reçoit le nom du fichier, le numéro de la ligne voulue, un drapeau pour affichage des messages de debug et renvoie un String correspondant à la ligne.
- Le principe consiste à défiler les caractères et à rechercher les saut de ligne (code ascii=10) et à extraire les caractères entre 2 sauts de lignes.

```
// fonction lecture de ligne dans un fichier de carte mémoire SD
// par X. HINAULT - www.mon-club-elec.fr -GPL v3 - Mars 2013

String getLine(char* nomFichierIn, int lineNumberIn, boolean debug) {
    // la fonction reçoit :
    // > le nom du fichier
    // > le numéro de la ligne - 1ère ligne = ligne N°1
    // > un drapeau pour messages debug
    // la fonction renvoie le String correspondant à la ligne

    File dataFileIn=SD.open(nomFichierIn, FILE_READ); // ouvre le fichier en lecture - NOM FICHIER en 8.3 ++++

    if (debug) Serial.println("-----");

    String strLine="";
    int comptLine=0;
    int lastPosition=0; // dernière position
    int currentPosition=0; // dernière position

    if (dataFileIn){ // le fichier est True si créé
        if (debug)Serial.println("Ouverture fichier OK");

        while (dataFileIn.available()) { // tant que des données sont disponibles dans le fichier
            // le fichier peut etre considéré comme un "buffer" de données comme le buffer du port série

            char c = dataFileIn.read(); // lit le caractère suivant
            //Serial.print(c); // affiche le caractère courant

            if (c==10) { // si le caractère est \n : c'est une fin de ligne
                comptLine=comptLine+1;

                if (comptLine==lineNumberIn) { // si on a atteint la ligne voulue
                    currentPosition=dataFileIn.position(); // mémorise la position actuelle

                    for (int i=lastPosition; i<currentPosition; i++) { // on défile de la dernière position retenue à la position courante
                        dataFileIn.seek(i); // se position en i
                        strLine=strLine+char(dataFileIn.read()); // ajoute le caractère au strLine
                    } // fin for
                    break; // sort du while
                } // fin if arrivé à la ligne voulue
                lastPosition=dataFileIn.position();
            } // fin if saut de ligne

        } // fin while available

        dataFileIn.close(); // fermeture du fichier obligatoire après accès

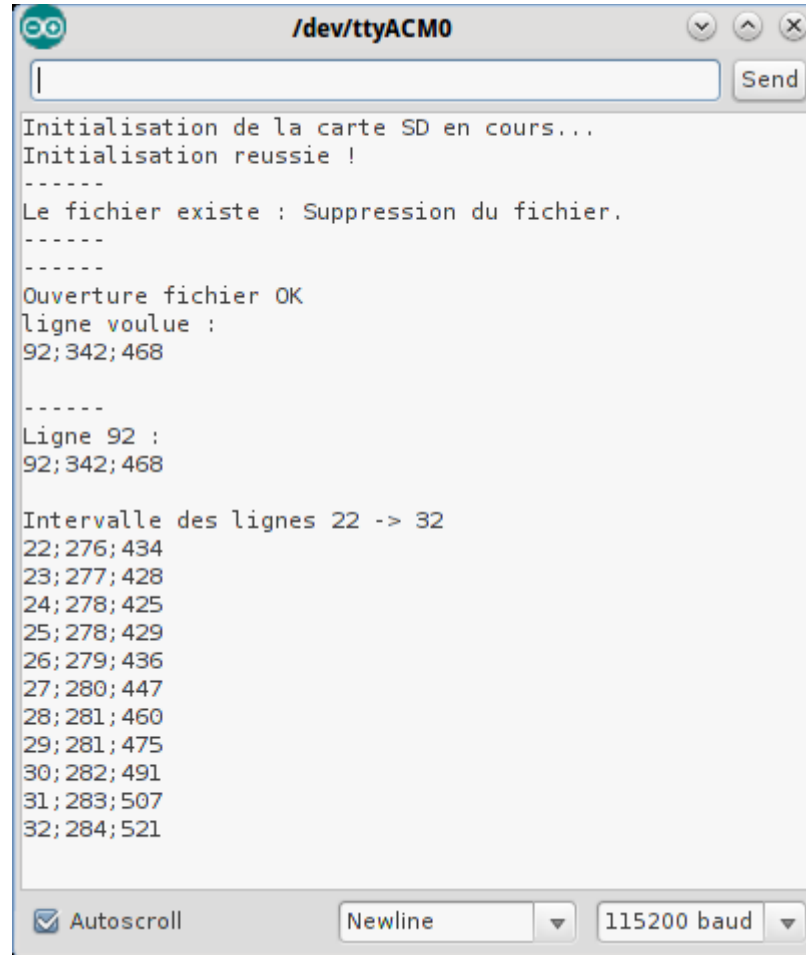
        if (debug) Serial.println("ligne voulue : ");
        if (debug) Serial.println(strLine);
        if (debug) Serial.println("-----");

    } // si fichier existe
    else { // sinon = si probleme creation
        Serial.println("Probleme creation fichier");
    } // fin else

    return(strLine);
} // fin fonction getLine
```

## Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 », ce qui donne ...



```
Initialisation de la carte SD en cours...
Initialisation reussie !
-----
Le fichier existe : Suppression du fichier.
-----
Ouverture fichier OK
ligne voulue :
92;342;468

-----
Ligne 92 :
92;342;468

Intervalle des lignes 22 -> 32
22;276;434
23;277;428
24;278;425
25;278;429
26;279;436
27;280;447
28;281;460
29;281;475
30;282;491
31;283;507
32;284;521
```

Et voilà, vous pouvez très simplement accéder à une ligne voulue. Cool non ?  
Sur le même principe, il est facile d'envisager une fonction pour compter les lignes... Allez, on continue... !



## 11. Carte SD : Obtenir des informations sur un fichier : taille, nombre de lignes.

### Ce qu'on va faire ici...

- Je vous propose à présent de faire quelque chose de plus simple, presque de la détente... ! On va voir ici comment obtenir des informations sur un fichier présent sur la carte SD.
- Tout d'abord, obtenir sa taille à l'aide de la fonction `size()` qui renvoie le nombre d'octets.
- Ensuite, nous verrons comment connaître le nombre de lignes contenues dans le fichier à l'aide d'une petite fonction que je vous propose.
- Pour l'essentiel, on reprend un programme déjà vu précédemment, avec l'ajout du code spécifique au niveau de la fonction `setup()`, donc vous êtes ici en terrain connu.
- Ce code est disponible ici : <https://gist.github.com/sensor56/f8c720215efdcfcdbdf2c>

### Entête déclarative

#### Inclusion des bibliothèques utiles

- On commence par inclure les bibliothèques
  - la bibliothèque Arduino **SD** qui permet d'utiliser une carte mémoire SD avec une carte Arduino
  - la bibliothèque Arduino **SPI** qui permet d'utiliser la communication série SPI, ici utilisée par la carte SD. Noter qu'il n'est pas indispensable d'inclure cette bibliothèque à ce niveau, ceci étant fait automatiquement par la bibliothèque SD, mais par mesure de clarté, j'ai pris l'habitude de le faire.

#### Déclaration broche utilisée

- On déclare la broche utilisée pour sélectionner la carte SD pour la communication SPI (si vous ne savez pas ce qu'est la communication SPI, voir le tuto dédié) : cette broche est variable selon les shields avec étage mémoire SD utilisés (voir la doc de votre shield au besoin) :
  - soit la broche 10 par défaut (cas du shield Mémoire de Snootlab par exemple)
  - soit la broche 4 pour le shield Ethernet

#### Déclaration des objets utiles

- **Noter que l'on ne déclare pas l'objet SD**, qui est déclaré implicitement dans la bibliothèque SD.

```
#include <SPI.h> // fichier librairie pour communication SPI utilisée par carte SD
#include <SD.h> // fichier librairie pour gestion de la carte SD

// broche de sélection de la carte SD : à adapter au shield utilisé
const int selectSD=10; // par défaut
//const int selectSD=4; // pour le shield Arduino
```

## Fonction **setup()**

### Configuration des broches numériques

- **Point important : même si cette broche n'est pas utilisée pour sélectionner la carte mémoire SD, il est impératif de mettre en sortie la broche /SS de la carte Arduino** (=broche 10 sur la UNO), sinon la communication SPI ne se fera pas correctement. C'est ce que nous faisons ici.

### Initialisation série

- On initialise la communication série à 115200 bauds

### Initialisation de la carte SD

- on affiche ensuite un message indiquant que l'initialisation de la carte SD est en cours
- puis on appelle la fonction **SD.begin()** en passant en paramètre la broche utilisée : noter que cette fonction est placée au sein d'une condition d'une boucle **if()** : ceci est possible car la fonction **SD.begin()** renvoie True si l'initialisation se passe bien et False dans le cas contraire.
  - si l'initialisation ne se réalise pas correctement (le ! inverse la condition) : alors on affiche le message d'échec,
  - puis on sort de la fonction **setup()** à l'aide de l'instruction **return**
- sinon, on se retrouve après la condition et on affiche un message de succès.

```
//--- fonction setup exécutée une fois au lancement
void setup(){

  // configuration des broches E/S
  pinMode(10, OUTPUT); // IMPORTANT ++ : laisser la broche /SS(=10) en sortie - obligatoire avec librairie SD

  // configuration de la communication série
  Serial.begin(115200); // utiliser le meme debit coté Terminal Serie

  //--- initialisation de la carte SD
  Serial.println("Initialisation de la carte SD en cours...");

  if (!SD.begin(selectSD)) { // si initialisation avec broche selectSD en tant que CS n'est pas réussie
    Serial.println("Echec initialisation!"); // message port Série
    return; // sort de setup()
  } // if SD begin()

  //--- si initialisation réussie : on se place ici :
  Serial.println("Initialisation reussie !"); // message port Série
} // fin setup
```

## Fonction **setup()** (suite)

### Test existence fichier et suppression si il existe déjà

- A présent, nous commençons par déclarer un tableau de `char` correspondant au nom du fichier. **IL FAUT VRAIMENT FAIRE ATTENTION ET BIEN VERIFIER à ce que le format du nom du fichier soit bien en « 8.3 »** c'est à dire 8 lettres maxi pour le nom et 3 lettres maxi après le « . » L'écriture sur carte SD avec la librairie Arduino SD fonctionne très bien... sous réserve de bien respecter cette règle. Sinon, vous obtiendrez des messages d'erreur ou bien les opérations sur les fichiers ne se feront pas correctement !
- Ensuite, on teste si le fichier existe avec la fonction `SD.exists()` en passant en paramètre le nom du fichier. Cette fonction renvoie `True/False`, c'est pourquoi il est possible de la passer en condition à un `if()` de manière à exécuter le code voulu uniquement si le fichier existe :
  - ici si le fichier existe, nous l'effaçons à l'aide de la fonction `SD.remove()`
  - des messages série permettent de suivre les opérations...
  - ce qui donne :

```
//---- test existence fichier et suppression du fichier si il existe ---
char nomFichier[]="testFile.txt"; // utiliser un nom de fichier format 8.3 - doit etre un tableau de char

if (SD.exists(nomFichier)) { // si le fichier existe

    Serial.println("-----");
    Serial.print("Le fichier existe : "); // affiche message
    SD.remove(nomFichier); // efface le fichier
    Serial.println("Suppression du fichier."); // affiche message

} // fin si fichier existe
```

## Fonction **setup()** (suite)

### Création du fichier

- On commence par créer un objet File représentant un fichier ou un répertoire, à l'aide de la fonction **SD.open()** en mode **FILE\_WRITE** : ouvre le fichier pour lecture et écriture, en démarrant au début du fichier. *Si le fichier est ouvert pour écriture, il sera créé si il n'existe pas déjà (cependant le chemin le contenant doit déjà exister).*

### Écriture de données dans le fichier

- Une fois l'objet **File** créé, il est possible de tester son existence directement à l'aide d'une condition **if()** : un objet **File** non vide sera un équivalent de **True**. C'est ce que l'on commence par faire.
- Si le fichier existe (c'est à dire si il a correctement été ouvert) :
  - on écrit des données dedans comme on le ferait sur le port Série ou le réseau, à l'aide des fonctions **println()** et **print()** tout simplement !! *Ici, on écrit quelques lignes dans le fichier... puisque l'on veut compter les lignes !*
  - puis, une fois terminé, on ferme le fichier avec l'instruction **close()** de l'objet **File**. Remarquer au passage que la fonction **open()** est une fonction de l'objet **SD** et que la fonction **close()** est une fonction de l'objet **File**.

```
// création nouveau fichier
File dataFile=SD.open(nomFichier, FILE_WRITE); // crée / ouvre un objet fichier et l'ouvre en mode écriture - NOM FICHER en 8.3 ++++
// Important : Si le fichier est ouvert pour écriture, il sera créé si il n'existe pas déjà (cependant le chemin le contenant doit déjà exister)

Serial.println("-----");

if (dataFile){ // le fichier est True si créé
  Serial.println("Creation nouveau fichier OK");
  dataFile.println("Ligne 1");
  dataFile.println("Ligne 2");
  dataFile.println("Ligne 3");
  dataFile.println("Ligne 4");
  dataFile.println("Ligne 5");
  dataFile.close(); // fermeture du fichier obligatoire après accès
} // si fichier existe
else { // sinon = si probleme creation
  Serial.println("Probleme creation fichier");
} // fin else
```

## Fonction **setup()** (suite)

### Connaître la taille d'un fichier

- Pour connaître la taille du fichier, c'est assez simple : il suffit d'utiliser la fonction `size()` de la classe `File` de la librairie SD. **Point important : il faut impérativement que le fichier soit OUVERT.**
- ce qui nous donne :

```
//----- connaitre la taille du fichier

dataFile=SD.open(nomFichier, FILE_READ); // ouvre le fichier en lecture - NOM FICHER en 8.3 ++++
// un seul fichier ne peut etre ouvert à la fois - fermer au préalable tout fichier déjà ouvert

Serial.println("-----");

Serial.print("Taille du fichier : ");
Serial.print(dataFile.size());
Serial.println(" octets.");

dataFile.close(); // fermeture du fichier
```

- Remarquer qu'il faut à chaque fois ouvrir le fichier, récupérer la taille et le fermer, d'où l'idée de rassembler l'ensemble dans une fonction qui ne reçoit que le nom du fichier et renvoie la taille selon (il suffira d'appeler alors la taille sous la forme `taille = getFileSize(nomFichier,true)`) :

```
// Fonction renvoyant la taille en octet d'un fichier sur carte mémoire SD
// par X. HINAULT - www.mon-club-elec.fr - GPLv3
int getFileSize(char* nomFichierIn, boolean debug) {
    // La fonction reçoit :
    // > le nom du fichier au format 8.3
    // > un drapeau d'affichage de messages de debug
    // la fonction renvoie le nombre d'octet du fichier

    File dataFileIn=SD.open(nomFichierIn, FILE_READ); // ouvre le fichier en lecture - NOM FICHER en 8.3 ++++

    if (debug) Serial.println("-----");
    Serial.println("-----");

    if (dataFileIn){ // le fichier est True si créé

        if (debug)Serial.println("Ouverture fichier OK");
        Serial.print("Taille du fichier : ");
        Serial.print(dataFileIn.size());
        Serial.println(" octets.");
        return(dataFileIn.size()); // renvoie la taille du fichier
        if (debug) Serial.println("-----");
        dataFileIn.close(); // fermeture du fichier obligatoire après accès
    } // si fichier existe
    else { // sinon = si probleme creation
        Serial.println("Probleme ouverture fichier");
        return(0);
    } // fin else
} // fin fonction getFileSize
```



## Fonction **setup()** (suite)

### Connaître le nombre de ligne

- Pour connaître le nombre de ligne, je vous propose une petite fonction pour le faire automatiquement et qui s'appelle `getNumberOfLines()` et qui renvoie le nombre de lignes à partir du nom du fichier. La fonction est décrite ci-après :
- Ce qui donne :

```
//----- connaître le nombre de lignes du fichier

int nombreLignes=getNumberOfLines(nomFichier, true); // fonction nombre de lignes du fichier

Serial.println("Nombre de lignes : ");
Serial.println(nombreLignes);
Serial.println("-----");

} // fin setup
```

## Fonction **loop()**

- Laissée vide ici

```
//--- fonction loop exécutée en boucle infinie
void loop(){

    // laissée vide ici

} // fin loop
```

## Fonction **getNumberOfLines()**

- Au niveau de cette fonction « maison », on commence par ouvrir le fichier, puis on défile tous les caractères en vérifiant la présence des sauts de ligne (ascii=10) : à chaque fois qu'un saut de ligne est détecté, on incrémente une variable de comptage. Au final, on obtient le nombre de lignes !

```
// Fonction renvoyant le nombre de lignes d'un fichier sur carte mémoire SD
// par X. HINAULT - www.mon-club-elec.fr - GPLv3 - Mars 20123

int getNumberOfLines(char* nomFichierIn, boolean debug) {
  // La fonction reçoit :
  // > le nom du fichier au format 8.3
  // > un drapeau d'affichage de messages de debug

  // la fonction renvoie le nombre de ligne

  File dataFileIn=SD.open(nomFichierIn, FILE_READ); // ouvre le fichier en lecture - NOM FICHER en 8.3 ++++

  if (debug) Serial.println("-----");

  int comptLine=0; // variable de comptage des lignes

  if (dataFileIn){ // le fichier est True si créé

    if (debug)Serial.println("Ouverture fichier OK");

    while (dataFileIn.available()) { // tant que des données sont disponibles dans le fichier
      // le fichier peut etre considéré comme un "buffer" de données comme le buffer du port série

      char c = dataFileIn.read(); // lit le caractère suivant
      if (debug)Serial.print(c); // affiche le caractère courant

      if (c==10) { // si le caractère est \n : c'est une fin de ligne
        comptLine=comptLine+1;
      } // fin if saut de ligne

    } // fin while available

    if (debug) Serial.println("-----");

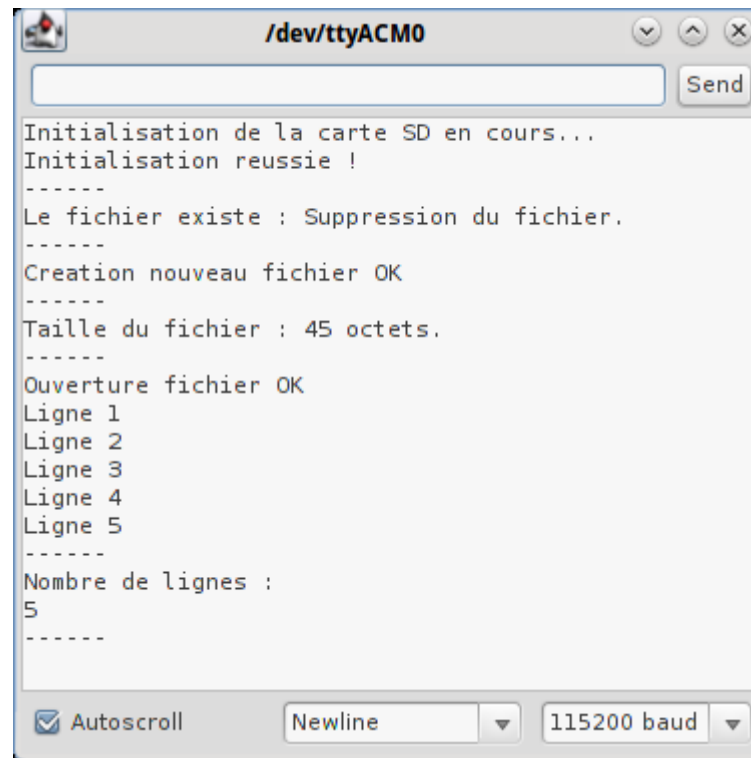
    dataFileIn.close(); // fermeture du fichier obligatoire après accès

  } // si fichier existe
  else { // sinon = si probleme creation
    Serial.println("Probleme ouverture fichier");
  } // fin else

  return(comptLine); // renvoie le nombre de ligne
} // fin fonction getNumberOfLines
```

## Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 », ce qui donne ...



```
/dev/ttyACM0
Initialisation de la carte SD en cours...
Initialisation reussie !
-----
Le fichier existe : Suppression du fichier.
-----
Creation nouveau fichier OK
-----
Taille du fichier : 45 octets.
-----
Ouverture fichier OK
Ligne 1
Ligne 2
Ligne 3
Ligne 4
Ligne 5
-----
Nombre de lignes :
5
-----
Autoscroll Newline 115200 baud
```

Tout bête, mais cela sera bien pratique à l'occasion !

## 12. Afficher le contenu d'une carte mémoire SD avec Arduino : le programme

### Ce qu'on va faire ici...

- A présent, je vous propose de voir comment afficher le contenu d'une carte SD, c'est à dire la liste de tous les répertoires et fichiers présents sur la carte SD. Ce code est basé sur l'exemple listFiles fournit avec la librairie SD... que j'adapte légèrement et en l'expliquant bien sûr.
- Une telle opération de lecture du contenu d'un fichier pourra sembler « banale » sur un ordinateur classique, mais ici, rappelons-le, nous n'utilisons qu'un simple micro-contrôleur (=Arduino) ce qui va donc nécessiter de coder à l'aide des fonctions simples de la librairie SD la lecture du contenu.
- Ce code est disponible ici : <https://gist.github.com/sensor56/1b1fd2ff2703bc42338b>

**ATTENTION** : nous allons lire ici le contenu d'un répertoire qui devra avoir été créé au préalable (sur votre ordinateur) avec quelques fichiers dedans.  
Dans ce code, ce répertoire s'appelle « /DOSSIER2 », mais ça pourra aussi être « / » si on veut lire le contenu de la carte SD.

### Entête déclarative

#### Inclusion des librairies utiles

- On commence par inclure les librairies
  - la librairie Arduino **SD** qui permet d'utiliser une carte mémoire SD avec une carte Arduino
  - la librairie Arduino **SPI** qui permet d'utilisation la communication série SPI, ici utilisée par la carte SD. Noter qu'il n'est pas indispensable d'inclure cette librairie à ce niveau, ceci étant fait automatiquement par la librairie SD, mais par mesure de clarté, j'ai pris l'habitude de le faire.

#### Déclaration broche utilisée

- On déclare la broche utilisée pour sélectionner la carte SD pour la communication SPI (si vous ne savez pas ce qu'est la communication SPI, voir le tuto dédié) : cette broche est variable selon les shields avec étage mémoire SD utilisés (voir la doc de votre shield au besoin) :
  - soit la broche 10 par défaut (cas du shield Mémoire de Snootlab par exemple)
  - soit la broche 4 pour le shield Ethernet

#### Déclaration des objets utiles

- **Noter que l'on ne déclare pas l'objet SD**, qui est déclaré implicitement dans la librairie SD.

```
#include <SPI.h> // fichier librairie pour communication SPI utilisée par carte SD
#include <SD.h> // fichier librairie pour gestion de la carte SD

// broche de sélection de la carte SD : à adapter au shield utilisé
const int selectSD=10; // par défaut
//const int selectSD=4; // pour le shield Arduino
```

## Fonction **setup()**

### Configuration des broches numériques

- **Point important : même si cette broche n'est pas utilisée pour sélectionner la carte mémoire SD, il est impératif de mettre en sortie la broche /SS de la carte Arduino** (=broche 10 sur la UNO), sinon la communication SPI ne se fera pas correctement. C'est ce que nous faisons ici.

### Initialisation série

- On initialise la communication série à 115200 bauds

### Initialisation de la carte SD

- on affiche ensuite un message indiquant que l'initialisation de la carte SD est en cours
- puis on appelle la fonction **SD.begin()** en passant en paramètre la broche utilisée : noter que cette fonction est placée au sein d'une condition d'une boucle **if()** : ceci est possible car la fonction **SD.begin()** renvoie True si l'initialisation se passe bien et False dans le cas contraire.
  - si l'initialisation ne se réalise pas correctement (le ! inverse la condition) : alors on affiche le message d'échec,
  - puis on sort de la fonction **setup()** à l'aide de l'instruction **return**
- sinon, on se retrouve après la condition et on affiche un message de succès.

```
//--- fonction setup exécutée une fois au lancement
void setup(){

  // configuration des broches E/S
  pinMode(10, OUTPUT); // IMPORTANT ++ : laisser la broche /SS(=10) en sortie - obligatoire avec librairie SD

  // configuration de la communication série
  Serial.begin(115200); // utiliser le meme debit coté Terminal Serie

  //--- initialisation de la carte SD
  Serial.println("Initialisation de la carte SD en cours...");

  if (!SD.begin(selectSD)) { // si initialisation avec broche selectSD en tant que CS n'est pas réussie
    Serial.println("Echec initialisation!"); // message port Série
    return; // sort de setup()
  } // if SD begin()

  //--- si initialisation réussie : on se place ici :
  Serial.println("Initialisation reussie !"); // message port Série
```

## Fonction **setup()** (suite)

### Accès au contenu d'un répertoire

- Grâce à la fonction `getContentDirSerial()` que je vous propose ci-après, il devient assez simple d'afficher le contenu d'une carte mémoire SD. La fonction reçoit simplement le nom du fichier et un paramètre fixant le nombre de tabulation initial à utiliser,
- ce qui donne :

```
//----- affiche le contenu du répertoire racine

Serial.println("Contenu de la carte SD : ");
char racineSD[] = "/"; // répertoire à ouvrir - "/" est la racine de la carte SD
getContentDirSerial(racineSD, 0); // affiche contenu d'un répertoire avec 0 tab au debut
// note : la fonction locale getContentDirSerial est ré-entrante dans elle-meme et affichera tous les sous-rép avec contenu

Serial.println();

//----- affiche le contenu d'un sous-répertoire
Serial.println("Contenu du dossier 2 : ");
getContentDirSerial("/DOSSIER2", 0); // affiche contenu d'un répertoire avec 0 tab au debut
// note : la fonction locale getContentDirSerial est ré-entrante dans elle-meme et affichera tous les sous-rép avec contenu
Serial.println();

Serial.println("-- Operation Terminee! -- ");

// tout fichier est accessible sous la forme absolue /chemin/nomfichier/
} // fin setup
```

## Fonction **loop()**

- Laissée vide ici

```
//--- fonction loop exécutée en boucle infinie
void loop(){

    // laissée vide ici

} // fin loop
```

## Fonction `getContentDirSerial()`

- Au niveau de cette fonction, on commence par ouvrir le fichier, puis on obtient le fichier suivant à l'aide de la fonction `openNextFile()` : si le fichier existe, on recommence, sinon, on ré-appelle la fonction elle-même en décalant d'une tabulation, ce qui aboutit à afficher tout le contenu.

```
// Fonction affichant le contenu d'un répertoire de la carte mémoire SD sur le port série – supporte appels itératifs de la fonction
// modifié par X. HINAULT - www.mon-club-elec.fr - GPLv3 - 03/2013
// d'après exemple listFiles de la librairie SD | created Nov 2010 by David A. Mellis | modified 9 Apr 2012 by Tom Igoe

void getContentDirSerial(char* dirIn, int numTabsIn) { // la fonction reçoit le nom du répertoire, le décalage tab et le drapeau de debug
  // La fonction reçoit :
  // > le nom du répertoire
  // > le nombre de tab de début - obligatoire car reçu en paramètre ré-entrant...
  // la fonction renvoie rien

  File openDir=SD.open(dirIn); // ouvre la SD Card à l'emplacement indiqué et renvoie l'objet File correspondant
  openDir.rewindDirectory(); // RAZ la position dans le rép

  boolean firstPass=false; // drapeau 1er passage while

  boolean debug=true;

  while(true) { // tant que vrai = crée une "loop" qui s'exécute tant que contenu
    // la sortie se fait par break;

    File contenu = openDir.openNextFile(); // ouvre le fichier ou repertoire suivant - NB : un répertoire est aussi un objet File

    if (! contenu) { // si aucun nouveau fichier /repertoire

      //if (debug) Serial.println("*** vide***");

      contenu.close(); // ferme fichier avant sortie while
      break; // sort de la fonction

    } // fin si aucun nouveau fichier / répertoire

    if (firstPass==false) { // si premier passage = on remonte pour etre sur d'etre au debut du rep

      contenu.rewindDirectory();
      firstPass=true; // est déjà passé 1 fois

    } // fin if first pass

    // affiche le nombre de tab voulu - 0 si racine, 1 si sous Rép, 2 si sous-sous rép, etc..
    for (int i=0; i<numTabsIn; i++) {if (debug) Serial.print('\t');}

    if (debug) Serial.print(contenu.name()); // affiche le nom du fichier/repertoire courant

    if (contenu.isDirectory()) { // si le fichier est un répertoire
      if (debug) Serial.println("/"); // affiche un slash
      contenu.close(); // ferme fichier avant sortie while
      getContentDirSerial(contenu.name(), numTabsIn+1); // affiche le contenu en décalant d'un tab - Fonction ré-entrante qui s'appelle elle-meme!!
    } // fin si le fichier est un répertoire

    else { // sinon affiche la taille - les fichiers ont une taille, pas les répertoires

      if (debug)Serial.print("\t\t"); // affiche de Tab de décalage
      if (debug)Serial.print(contenu.size(), DEC); // affiche la taille
      if (debug)Serial.println(" octets"); // affiche la taille

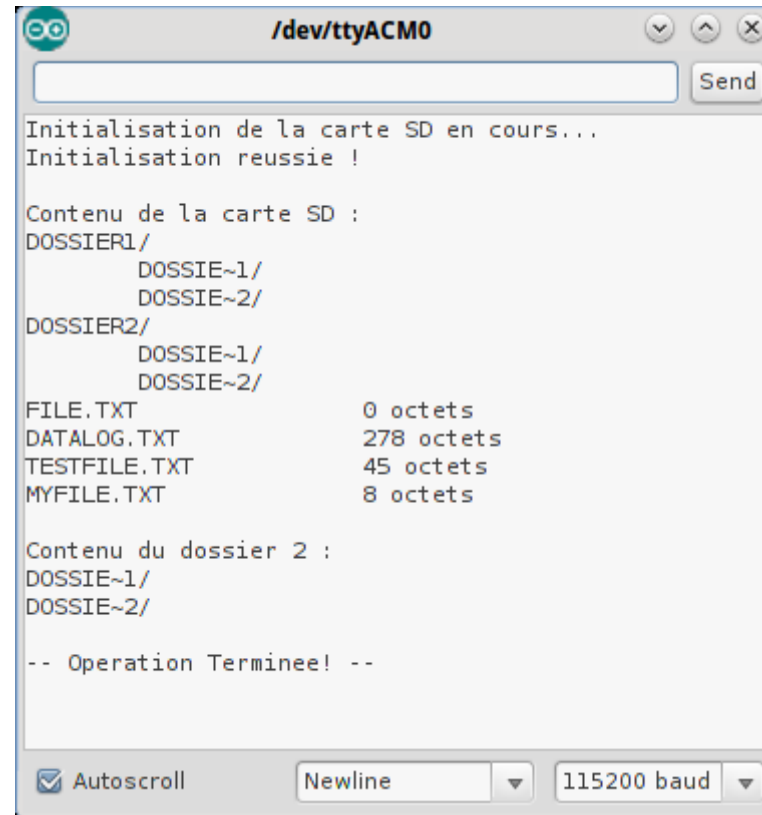
      contenu.close(); // ferme fichier avant sortie while

    } // fin sinon = si pas un rép
  } // fin while(true)
  openDir.close(); // ferme fichier avant sortie fonction
} // fin fonction getContentDirSerial
```



## Fonctionnement du programme

- Une fois la carte Arduino programmée, ouvrir le Terminal Série en réglant sur « newline » et « 115200 », ce qui donne ...



```
/dev/ttyACM0
Initialisation de la carte SD en cours...
Initialisation reussie !

Contenu de la carte SD :
DOSSIER1/
    DOSSIE~1/
    DOSSIE~2/
DOSSIER2/
    DOSSIE~1/
    DOSSIE~2/
FILE.TXT          0 octets
DATALOG.TXT       278 octets
TESTFILE.TXT      45 octets
MYFILE.TXT        8 octets

Contenu du dossier 2 :
DOSSIE~1/
DOSSIE~2/

-- Operation Terminee! --
```

Voilà, vous êtes en mesure de visualiser le contenu de la carte SD : cool non ?

**Remarquer que seuls les noms de répertoires sont affichés en 8 caractères, le ~ remplaçant les caractères supplémentaires.**

## 13. Usage avancé : Obtenir des infos sur la SD Card.

### Bon à savoir

- La librairie SD est basée sur une librairie appelée SdFatLib qui a été intégrée au projet Arduino dans un deuxième temps... Mais ce qu'il faut savoir, c'est que toutes les classes natives de cette librairie sont accessibles depuis le code Arduino : ceci permet l'utilisation de fonctions avancées parfois utiles.
- La documentation de la librairie SdFatLib est ici : [http://www.mon-club-elec.fr/pmwiki\\_reference\\_arduino/pmwiki.php?n=Main.LibrairieSdFatLib](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.LibrairieSdFatLib)
- L'exemple CardInfo notamment fourni avec la librairie SD vous montre comment utiliser ces fonctions natives de la librairie SdFatLib. Une fois n'est pas coutume, je vous mets ce code en copie « as is » pour information car il s'agit d'un usage avancé à connaître mais pas à maîtriser en pratique :
- Ce code est disponible ici : <https://gist.github.com/sensor56/9d029e9130a68d73cb8a>

```
// par X. HINAULT - licence GPL v3 - www.mon-club-elec.fr
// ateliers Arduino
// reprise exemple CardInfo de la librairie SD

// 13. Usage avancé : Obtenir des infos sur la SD Card.

#include <SPI.h> // fichier librairie pour communication SPI utilisée par carte SD
#include <SD.h> // fichier librairie pour gestion de la carte SD

// Création des objets racines donnant accès aux fonctions natives de la librairie SD
Sd2Card card;
SdVolume volume;
SdFile root;

// broche de sélection de la carte SD : à adapter au shield utilisé
//const int selectSD=10; // par défaut
const int selectSD=4; // pour le shield Arduino

//--- fonction setup exécutée une fois au lancement
void setup(){

    // configuration des broches E/S
    pinMode(10, OUTPUT); // IMPORTANT ++ : laisser la broche /SS(=10) en sortie - obligatoire avec librairie SD

    // configuration de la communication série
    Serial.begin(115200); // utiliser le meme debit coté Terminal Serie

    //--- initialisation de la carte SD
    Serial.println("Initialisation de la carte SD en cours...");

    if (!card.init(SPI_HALF_SPEED, selectSD)) {
        Serial.println("Echec initialisation!"); // message port Série
        return; // sort de setup()
    } // if echec init

    //--- si initialisation réussie : on se place ici :
    Serial.println("Initialisation reussie !"); // message port Série

    // affichage du type de la carte
```

```

Serial.print("\nType de la carte SD : ");
switch(card.type()) {
  case SD_CARD_TYPE_SD1:
    Serial.println("SD1");
    break;
  case SD_CARD_TYPE_SD2:
    Serial.println("SD2");
    break;
  case SD_CARD_TYPE_SDHC:
    Serial.println("SDHC");
    break;
  default:
    Serial.println("Inconnu");
}

//Essai d'ouverture du répertoire racine / : la partition doit etre FAT 32 ou FAT 16
if (!volume.init(card)) {
  Serial.println("Ne trouve pas de partition FAT16/FAT32.\nFormater la carte avec le bon format");
  return;
}

// Affiche la taille et le type de la premiere partition
uint32_t volumesize;
Serial.print("\nVolume de type FAT");
Serial.println(volume.fatType(), DEC);
Serial.println();

volumesize = volume.blocksPerCluster(); // les clusters sont des collections de blocs
volumesize *= volume.clusterCount();
volumesize *= 512; // les blocs SD card blocks comportent 512 octets
Serial.print("Taille volume (octets): ");
Serial.println(volumesize);
Serial.print("Taille volume (Ko): ");
volumesize /= 1024;
Serial.println(volumesize);
Serial.print("Taille volume (Mo): ");
volumesize /= 1024;
Serial.println(volumesize);

Serial.println("\nListe des fichiers presents sur la carte(nom, date et taille en octets): ");
root.openRoot(volume);

// liste tous les fichiers presents sur la carte
root.ls(LS_R | LS_DATE | LS_SIZE);
} // fin setup

//--- fonction loop exécutée en boucle infinie
void loop(){
  // laissée vide ici
} // fin loop

```

## 14. Les éléments du langage Arduino étudiés dans cet atelier

### Fonctions de la librairie SD

#### *Classe SD (accès à la carte SD, manipulation des fichiers et répertoires)*

- `begin()` : initialise la carte SD avec la broche voulue – Renvoie True/False
- `exists()` : teste si un répertoire ou un fichier existe sur la carte SD
- `mkdir()` : crée un répertoire
- `open()` : ouvre ou crée un fichier en écriture
- `remove()` : efface un fichier
- `rmdir()` : efface un répertoire

#### *Classe File (lecture/écriture dans des fichiers individuels)*

- `available()` : teste si octets disponibles en lecture
- `close()` : ferme le fichier
- `flush()` : écrit physiquement les données dans le fichier
- `peek()` : lit un octet dans le fichier sans passer à l'octet suivant
- `position()` : renvoie position courante au sein du fichier
- `print()` : écrit les données dans le fichier sans saut de ligne
- `println()` : écrit les données dans le fichier avec saut de ligne
- `seek()` : se place à la position voulue
- `size()` : renvoie la taille du fichier en nombre d'octets
- `read()` : lit un octet dans un fichier
- `write()` : écrit un octet dans un fichier
- `isDirectory()` : test si un fichier est un répertoire
- `openNextFile()` : ouvre le fichier suivant
- `rewindDirectory()` : se place au niveau du premier fichier/répertoire

La documentation complète du langage Arduino en français est disponible ici :  
[http://www.mon-club-elec.fr/pmwiki\\_reference\\_arduino/pmwiki.php?n=Main.ReferenceMaxi](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.ReferenceMaxi)

## **15. A présent, vous devriez être capable :**

- d'extraire une ligne ou un groupe de lignes voulus au sein d'un fichier de données sur une carte SD
- d'obtenir des informations sur un fichier : taille, nombre de lignes,
- d'afficher le contenu d'une carte mémoire SD avec Arduino
- de présenter un usage avancé : obtenir des infos sur la carte SD.

## Table des matières

Mémorisation de données sur une carte mémoire SD avec Arduino : Obtenir des informations sur les fichiers et les répertoires.

Intro |

Matériel nécessaire pour les ateliers Arduino |

Matériel spécifique nécessaire pour cet atelier |

Pour info : les cartes mémoires SD |

Introduction à la manipulation de fichiers |

Rappel : Langage Arduino : Introduction aux bibliothèques |

La bibliothèque Arduino SD |

Utiliser une carte mémoire SD avec Arduino : Préparatifs |

Utiliser une carte mémoire SD avec Arduino : le montage |

Extraire une ligne ou un groupe de lignes voulus au sein d'un fichier de données sur une carte SD |

Carte SD : Obtenir des informations sur un fichier : taille, nombre de lignes. |

Afficher le contenu d'une carte mémoire SD avec Arduino : le programme |

Usage avancé : Obtenir des infos sur la SD Card. |

Les éléments du langage Arduino étudiés dans cet atelier |

A présent, vous devriez être capable : |

**Bravo !**  
vous avez terminé cet atelier Arduino !



Prêt pour la suite ? Retrouvez de nombreux autres thèmes d'ateliers Arduino ici :

[http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.ATELIERS](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ATELIERS)