

Mémorisation de données sur une carte mémoire SD avec Arduino : Apprendre à utiliser les fichiers et les répertoires et à écrire des données.



Ateliers Arduino

par X. HINAULT

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

Pour obtenir tout autres types de licence d'utilisation (enseignement, commercial, etc...), veuillez contacter l'auteur ici : 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

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 !

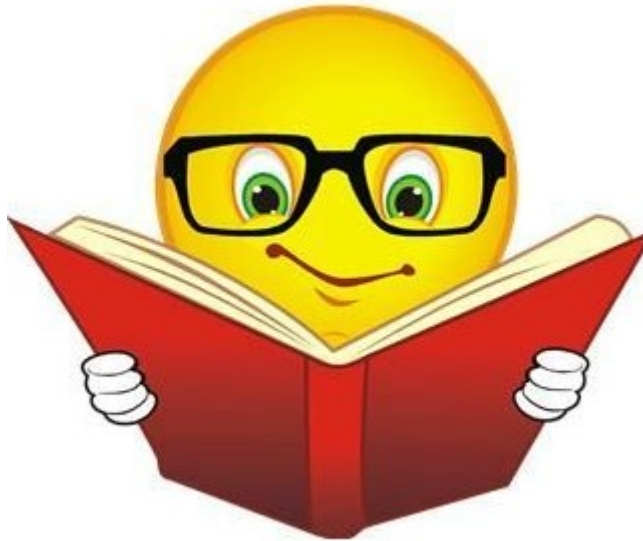
Lancer également le logiciel Arduino et connecter votre carte Arduino afin de pouvoir tester au fur et à mesure les codes d'exemples !

1. Intro

L'objectif ici est :

- d'apprendre à utiliser une carte mémoire SD avec Arduino
- d'apprendre à initialiser la carte mémoire SD
- d'apprendre à créer un fichier et écrire des données dedans
- d'apprendre à lire des données dans un fichier sur carte mémoire SD
- d'apprendre à écrire une série de données dans un fichier et les visualiser dans le Terminal Série

... afin d'être en mesure de maîtriser les bases de l'utilisation d'une carte mémoire SD avec Arduino.



Prêt ? C'est parti !

Comme vous allez le constater, le langage Arduino va vous permettre d'accéder aux données au sein d'une carte mémoire SD de la même façon que vous enverriez des données sur le port série : souplesse maximale !

D'autres part, il sera possible de donner à vos données le format de votre choix, notamment pour séparer les données, ce qui permettra d'utiliser ensuite directement le fichier dans un tableur par exemple !

Que du bon... et même du très bon !!

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

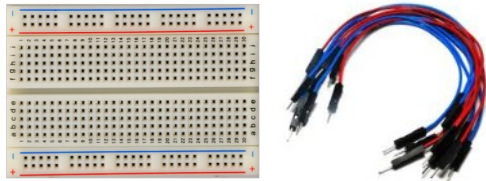


L'espace de développement Arduino associe :

- 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

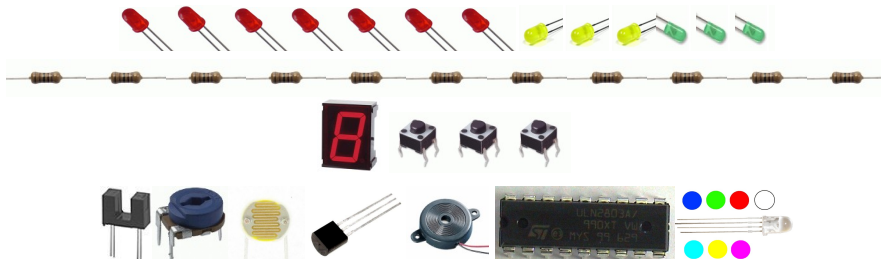


Pour réaliser des montages sans soudure, vous aurez besoin :

- 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

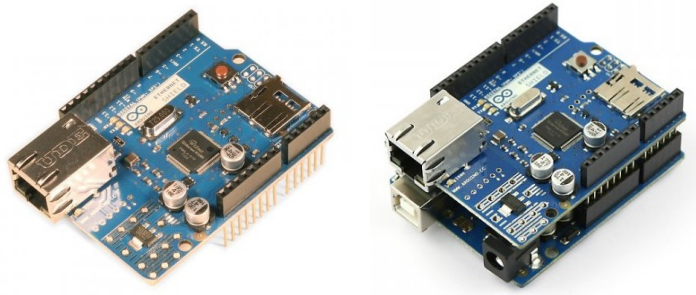
Pour les ateliers Arduino niveau débutant, vous devrez idéalement disposer des composants suivants :

- 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...

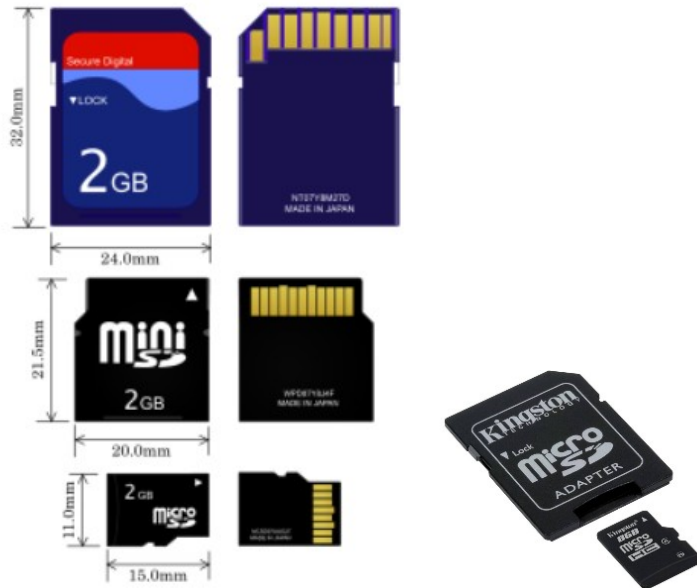
4. Pour info : les cartes mémoires SD

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 informatique, 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 bibliothèques

C'est quoi une bibliothèque 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 bibliothèques.

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 bibliothèque dédiée correspondante.

Principe général d'utilisation

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

```
#include <nombibliothèque.h> // bibliothèque 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 bibliothèque. Cet objet est en fait une instance (= un exemplaire) d'une Classe (=le moule) qui regroupe les fonctions de la bibliothèque. 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 bibliothèque 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 bibliothèques, mais grosso-modo, ça fonctionne de cette façon pour la plupart des bibliothèques Arduino.

Vous avez déjà utilisé une bibliothèque !

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 bibliothèque 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 bibliothèques standards Arduino

Les bibliothèques 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 bibliothèques standards du langage Arduino (**le logiciel donne la liste...**) :

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

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

- [La bibliothèque 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)

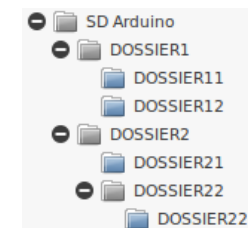


- Il existe également des lecteurs de carte SD externes à connecter sur port USB



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. Initialiser la carte mémoire SD : le programme

Ce qu'on va faire ici...

- Nous allons commencer par quelque chose de très simple : initialiser la carte mémoire SD, tout simplement.
- Le code est disponible ici : <https://gist.github.com/sensor56/58bdc2b8daea57dfa3ef>

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 bibliothèque pour communication SPI utilisée par carte SD
#include <SD.h> // fichier bibliothèque 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 **loop()**

- Laissée vide ici

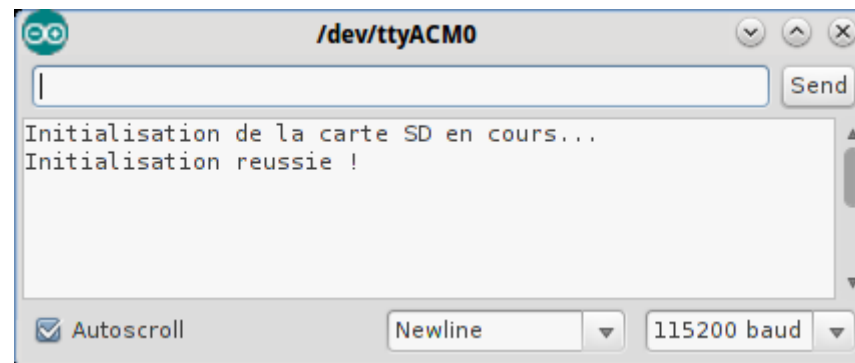
```
//--- fonction loop exécutée en boucle infinie
void loop(){

    // laissée vide ici

} // fin loop
```

Fonctionnement du programme

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



Tout bête, mais au moins vous êtes sûr que ça fonctionne !
La carte SD est bien initialisée... : les bases sont posées !

11. Carte SD : Créer un fichier et écrire des données dedans

Ce qu'on va faire ici...

- Maintenant que nous savons initialiser la carte mémoire SD, nous allons pouvoir créer un fichier et écrire des données dedans. Voyons cela...
- Le code est disponible ici : <https://gist.github.com/sensor56/cecfaa8220d0b3275903>

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 libairie 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()**. Cette fonction reçoit en paramètre :
 - le nom du fichier
 - et un argument qui précise le mode d'ouverture du fichier :
 - soit en mode **FILE_READ** : ouvre le fichier pour lecture, en démarrant au début du fichier
 - soit 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).**

Ecriture 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 !!
 - 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**.

Important à retenir :

Un seul fichier ne peut être ouvert à la fois = fermer au préalable tout fichier déjà ouvert.

```
// 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 ++++
// un seul fichier ne peut etre ouvert à la fois - fermer au préalable tout fichier déjà ouvert
// un fichier peut etre ouvert :
// > soit en mode FILE_READ: ouvre le fichier pour lecture, en démarrant au début du fichier.
// > soit en mode : FILE_WRITE: ouvre le fichier pour lecture et écriture, en démarrant au début du fichier.
// 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("Coucou");
  dataFile.close(); // fermeture du fichier obligatoire après accès
} // si fichier existe
else { // sinon = si probleme creation
  Serial.println("Probleme creation fichier");
} // fin else

} // fin setup
```

Remarquer une nouvelle fois la grande force du langage Arduino : les mêmes fonctions servent à écrire sur le port série, le réseau, la carte SD, etc... !!

Fonction **loop()**

- Laissée vide ici

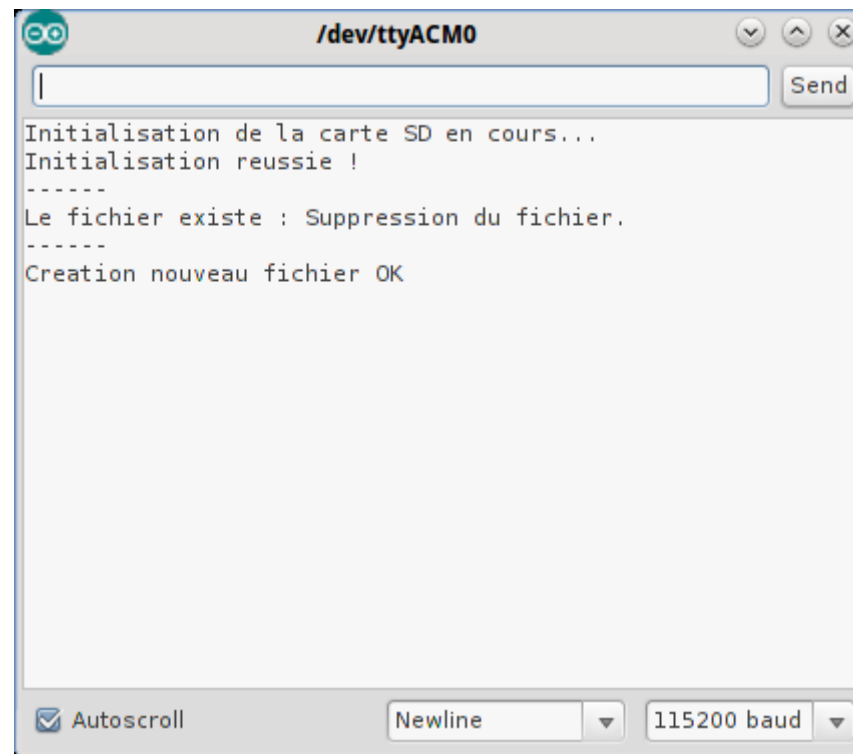
```
//--- fonction loop exécutée en boucle infinie
void loop(){

    // laissée vide ici

} // fin loop
```

Fonctionnement du programme

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



Votre premier fichier SD avec des données mémorisées sur une carte SD avec Arduino : cool non ?
A présent voyons comment lire ce fichier...

12. Carte SD : Lire des données dans un fichier sur carte mémoire SD

Ce qu'on va faire ici...

- Je reprends ici le programme précédent à l'identique, et au niveau de la fonction `setup()`, nous ajoutons le code permettant de lire le contenu du fichier.
- Le code est disponible ici : <https://gist.github.com/sensor56/cb40c5919da71904d9c3>

Fonction `setup()` (ajout)

Lecture des données dans le fichier

- On commence par ouvrir le fichier, mais cette fois en mode lecture, avec le paramètre `FILE_READ` passé à la fonction `open()`
- Pour la suite, si vous avez étudié le tuto dédié à la réception de caractères sur le port série, vous allez vite retomber sur vos pieds :
 - on commence par déclarer un `String` qui va nous servir à stocker le contenu du fichier
 - si le fichier existe (c'est à dire, si il a bien pu être ouvert en écriture avec `open()`) :
 - on réalise une boucle `while()` avec en condition la fonction `available()` de l'objet `File` qui permet de tester si des caractères sont disponibles (comme pour le port série)
 - ensuite, on lit le caractère suivant avec la fonction `read()` de l'objet `File` (toujours comme pour le port série...) et on ajoute ce caractère à notre `String`
 - une fois sorti du `while()`, c'est à dire lorsque tous les caractères ont été lus, **on n'oublie pas de fermer le fichier** avec la fonction `close()`
 - on affiche également divers messages, les caractères et bien sûr notre chaîne contenant le contenu du fichier :

```
// lecture du contenu 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("-----");

String contenuFichier=""; // chaine pour stocker contenu fichier

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

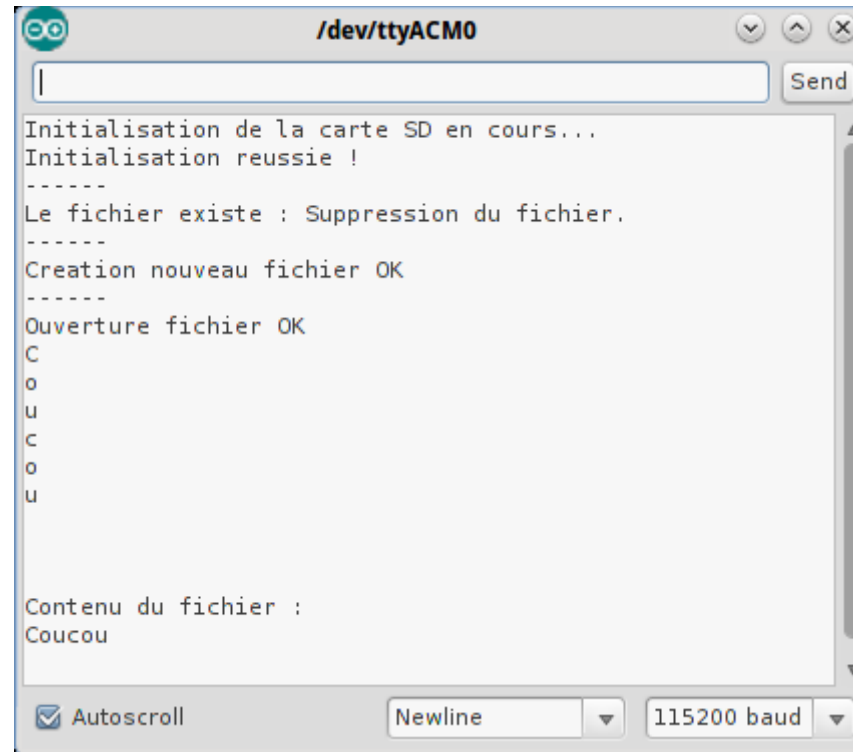
  while (dataFile.available()) { // tant que des données sont disponibles dans le fichier
    // le fichier peut etre considéré comme un "buffer" de données comme le buffer du port série
    char c = dataFile.read(); // lit le caractère suivant
    Serial.println(c); // affiche le caractère courant
    contenuFichier=contenuFichier+c; // ajoute le char au String
  } // fin while available

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

  Serial.println("Contenu du fichier : ");
  Serial.println(contenuFichier); // affiche le contenu du fichier
} // si fichier existe
else { // sinon = si probleme creation
  Serial.println("Probleme creation fichier");
} // fin else
```

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
-----
Ouverture fichier OK
C
o
u
c
o
u

Contenu du fichier :
Coucou

Autoscroll Newline 115200 baud
```

Bravo ! Vous avez réalisé votre premier enregistrement de données sur carte SD avec Arduino et réalisé la lecture des données !

Vous voyez, ce n'est pas si sorcier, à condition d'être un minimum rigoureux....



13. Carte SD : Ecrire une série de données dans un fichier et les visualiser dans le Terminal Série : le programme

Ce qu'on va faire ici...

- Bien, à présent nous allons poser les bases de l'enregistrement de série de données (appelé aussi « datalogging ») : il s'agit là de l'application probablement la plus utile d'une carte mémoire SD avec Arduino. Je vous propose pour commencer d'enregistrer tout simplement une série de résultats d'une mesure sur une voie analogique, disons ici une centaine de mesures.
- Un point important lorsque l'on enregistre des données, c'est de les « horodater », autrement indiquer le moment où la mesure a été réalisée. La façon la plus simple de le faire est d'utiliser valeur de la fonction `millis()` au moment de la mesure en même temps que la mesure, sous la forme `valeurMillis()` ; `valeurCAN`
- Il est également utile d'enregistrer un index de ligne avec la mesure (à des fins de vérification notamment) : comme nous allons le voir, en mettant une mesure par ligne, il sera possible d'accéder à la mesure voulue en précisant sa ligne, et le numéro d'index permettra d'être sûr d'accéder à la bonne ligne.

Par la suite, dans un autre tuto, nous verrons comment faire beaucoup mieux que ça, notamment en utilisant une fonction d'horloge « temp-réel »... mais pour le moment, nous utiliserons `millis()`, par souci de simplicité... « Trop d'infos... tue l'info ! »

- Le code de ce tuto est disponible ici : <https://gist.github.com/sensor56/56c172a26e0ff70ecc14>

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 bibliothèque pour communication SPI utilisée par carte SD
#include <SD.h> // fichier bibliothèque 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("---Acquisition en cours : veuillez patienter ---");

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 A3 + 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 des données dans le fichier

- On commence par ouvrir le fichier, mais cette fois en mode lecture, avec le paramètre `FILE_READ` passé à la fonction `open()`
- Pour la suite, si vous avez étudié le tuto dédié à la réception de caractères sur le port série, vous allez vite retomber sur vos pieds :
 - à la différence du programme précédent, nous n'utilisons pas de `String` ici, car lorsque le fichier devient trop important, il est impossible de le stocker dans un `String` au risque de saturer la RAM et de bloquer l'exécution du code. Nous n'utiliserons donc que des `char` pour l'affichage.
 - si le fichier existe (c'est à dire, si il a bien pu être ouvert en écriture avec `open()`) :
 - on réalise une boucle `while()` avec en condition la fonction `available()` de l'objet `File` qui permet de tester si des caractères sont disponibles
 - ensuite, on lit le caractère suivant avec la fonction `read()` de l'objet `File` et on affiche ce caractère.
 - une fois sorti du `while()`, c'est à dire lorsque tous les caractères ont été lus, **on n'oublie pas de fermer le fichier** avec la fonction `close()`
 - on affiche également divers messages si besoin :

```
// lecture du contenu du fichier
dataFile=SD.open(nomFichier, FILE_READ); // ouvre le fichier en lecture - 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
  Serial.println("Ouverture fichier OK");

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

  } // fin while available

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

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

} // fin setup
```

Fonction **loop()**

- Laissée vide ici

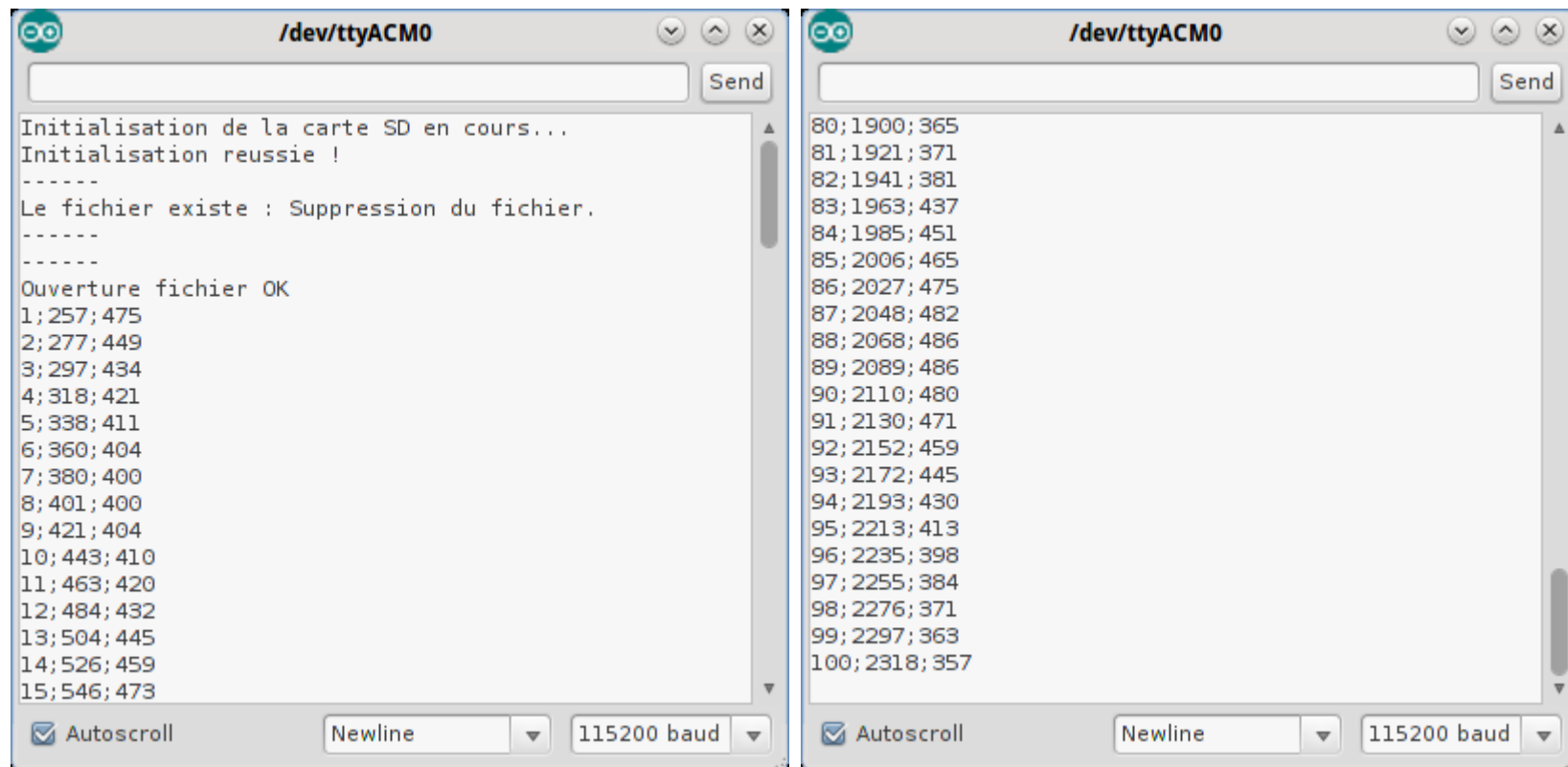
```
//--- fonction loop exécutée en boucle infinie
void loop(){

    // laissée vide ici

} // fin loop
```

Fonctionnement du programme

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

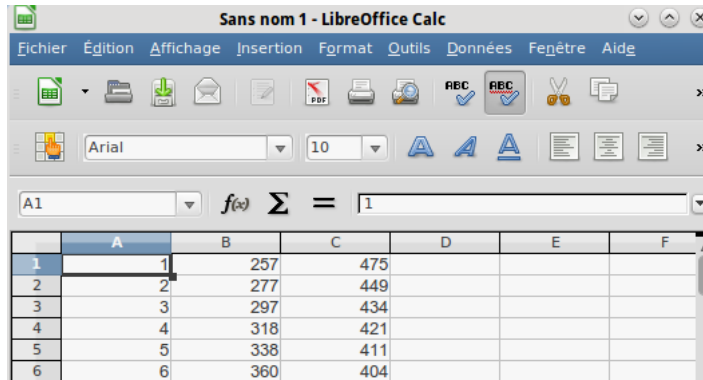


Bravo ! Vous venez de réussir votre premier enregistrement d'une série de mesures analogiques sur une carte mémoire SD...

Remarquer qu'il est aussi facile d'enregistrer 100 valeurs que 10 000 !

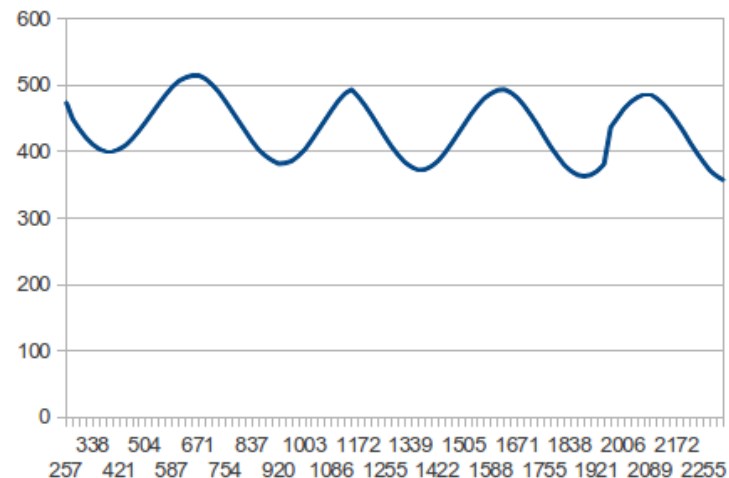
Truc : afficher les données dans un tableur !

- Comme nous avons séparé nos données par un ; et que nous les avons mises sur une seule ligne, elles sont très facilement utilisables dans un tableur !
- Il suffit de copier/coller les valeurs à partir du Terminal série ici, (ou bien en ouvrant le fichier enregistré sur la carte SD sur votre ordinateur : enlever la carte SD de son emplacement dans le shield et la connecter dans votre lecteur de carte SD)...
- ... puis de les coller dans le tableur à l'aide de menu Edition > collage spécial > texte non formaté
- et choisir le séparateur ;
- vous obtenez alors les données dans le tableur :



	A	B	C	D	E	F
1	1	257	475			
2	2	277	449			
3	3	297	434			
4	4	318	421			
5	5	338	411			
6	6	360	404			

- Et c'est alors un jeu d'enfant d'en extraire un graphique à partir des 2 colonnes « millis » et « valeur » ce qui donne dans mon cas :



Vous imaginez très bien je pense le potentiel de cette méthode : vous enregistrez des valeurs sur la carte SD pendant une longue période, puis vous récupérez les données ensuite dans un tableur très simplement pour les afficher sous forme de graphique de votre choix dans un banal tableur : polyvalence totale !!

14. Les éléments du langage Arduino étudiés dans cet atelier

Les 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

15. A présent, vous devriez être capable :

- d'utiliser une carte mémoire SD avec Arduino
- d'initialiser la carte mémoire SD
- de créer un fichier et écrire des données dedans
- de lire des données dans un fichier sur carte mémoire SD
- d'écrire une série de données dans un fichier et les visualiser dans le Terminal Série

Table des matières

Mémorisation de données sur une carte mémoire SD avec Arduino : Apprendre à utiliser les fichiers et les répertoires et à écrire des données.

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 |

Initialiser la carte mémoire SD : le programme |

Carte SD : Créer un fichier et écrire des données dedans |

Carte SD : Lire des données dans un fichier sur carte mémoire SD |

Carte SD : Ecrire une série de données dans un fichier et les visualiser dans le Terminal Série : le programme |

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