Lire et écrire des données sur une carte SD avec une carte Arduino / Genuino

De quoi s'amuser pendant un moment

https://www.carnetdumaker.net/articles/lire-et-ecrire-des-donnees-sur-une-carte-sd-avec-une-carte-arduino-genuino/#avant-propos-shields-sd-port-spi-et-cablage

par skywodd | fév. 4, 2017 | Licence (voir pied de page)

Catégories : Tutoriels Arduino | Mots clefs : Arduino Genuino Carte SD SD card sdfatlib

Cet article a été modifié pour la dernière fois le mars 14, 2017 à 5:51 après-midi

Cet article n'a pas été mis à jour depuis un certain temps, son contenu n'est peut être plus d'actualité.

Dans ce tutoriel, nous allons voir ensemble comment stocker des données sur une carte SD avec une carte Arduino / Genuino. On verra comment choisir, préparer et initialiser la carte SD. On verra ensuite comment manipuler les fichiers / dossiers qu'elle contient. En bonus, je vous propose quelques petits extraits de code pour vous faciliter la vie durant la conception de vos programmes.

Sommaire

- Avant propos : shields SD, port SPI et câblage
- Choix de la carte SD
- Préparation de la carte SD
- Montage de test
- Initialisation de la carte SD
- Opérations de haut niveau
    - Vérifier la présence d'un fichier ou d'un dossier
    - Créer un dossier ou un sous-dossier
    - Supprimer un dossier ou un sous-dossier
    - Supprimer un fichier
    - Ouvrir un fichier pour lecture ou écriture
    - Opérations possibles sur un fichier
        - Obtenir la taille en octets du fichier
        - Compter le nombre d'octets restants à lire
        - Lire un certain nombre d'octets
        - Écrire un certain nombre d'octets
        - Écrire du texte, des chiffres, etc.
        - Obtenir et / ou modifier la position actuelle dans le fichier
        - Forcer l'écriture immédiate des données en attente
        - Fermer le fichier
    - Opérations possibles sur un dossier
        - Vérifier si le dossier est bien un dossier
        - Ouvrir le fichier suivant dans le dossier
        - Revenir au premier fichier du dossier
        - Fermer le dossier
- Bonus : Lister tous les fichiers dans un dossier
- Bonus : Surveiller la mémoire RAM disponible
- Conclusion

Bonjour à toutes et à tous !

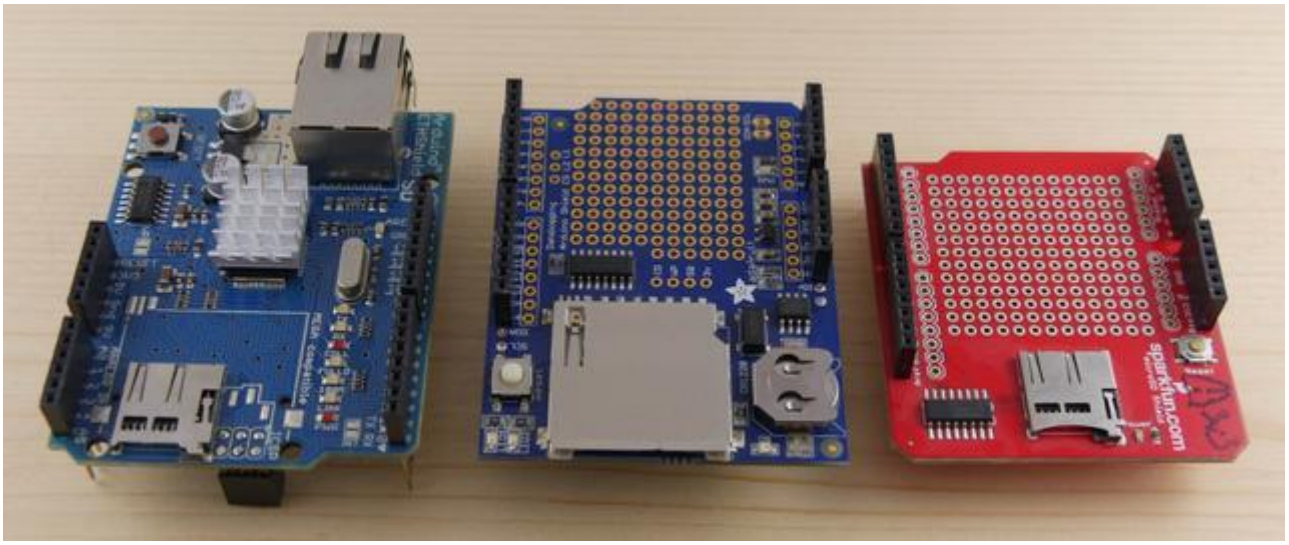
On a vu dans un précédent tutoriel comment stocker des données dans la mémoire interne d'une carte Arduino. Malheureusement, cette mémoire a une taille de quelques centaines d'octets seulement. Cela est suffisant pour certaines applications, mais pas pour d'autres.

La problématique du stockage de données est un sujet récurrent en développement Arduino. Qu'il s'agisse de stockage de mesures, de journaux d'événements ou simplement de fichiers de configuration, on a souvent besoin de stocker des données pour une utilisation ultérieure.

Comme toujours en électronique, plusieurs solutions sont possibles. On peut par exemple utiliser une mémoire EEPROM ou Flash, mais cela a un coût non négligeable et ce n'est pas très pratique. Une solution plus simple d'utilisation existe : utiliser une carte SD. C'est justement le sujet de l'article d'aujourd'hui.

Avant propos : shields SD, port SPI et câblage

Avant de commencer, parlons un peu des cartes disponibles et de leurs fonctionnements.



Diverses shields SD

Il existe dans le commerce de nombreuses cartes filles compatibles Arduino disposant d'un emplacement pour carte SD ou micro SD. On appelle couramment ces cartes filles des "shields".

La photographie ci-dessus montre trois shields SD classiques que l'on peut trouver un peu partout. La plus connue est sans aucun doute la shield Arduino Ethernet officielle (carte bleue à gauche). Les shield SD de Adafruit (au centre) et de Sparkfun (à droite) sont deux autres grands noms dans le milieu Arduino. Pour les amateurs d'eBay, n'ayez crainte, vous trouverez votre bonheur sans soucis

Techniquement, toutes ces shields font usage des broches `D11`, `D12` et `D13`. Ces trois broches forment un bus SPI. C'est via ce bus SPI que le microcontrôleur de la carte Arduino communique avec la carte SD.

PS Les shields compatibles Mega, comme la shield Ethernet officielle, utilisent les broches `D50`, `D51` et `D52` quand elles sont connectées à une carte Arduino Mega. Il est très simple de savoir si une shield SD est compatible Mega. Il suffit de regarder si la shield en question dispose du petit connecteur 2x3 broches que l'on peut voir juste à côté du connecteur micro SD de la carte bleue.

Petite parenthèse sur le bus SPI

Un bus SPI est un bus de communication série synchrone (avec horloge). Il y a toujours un maître (la carte Arduino) et un ou plusieurs esclaves (au moins une carte SD dans notre cas).

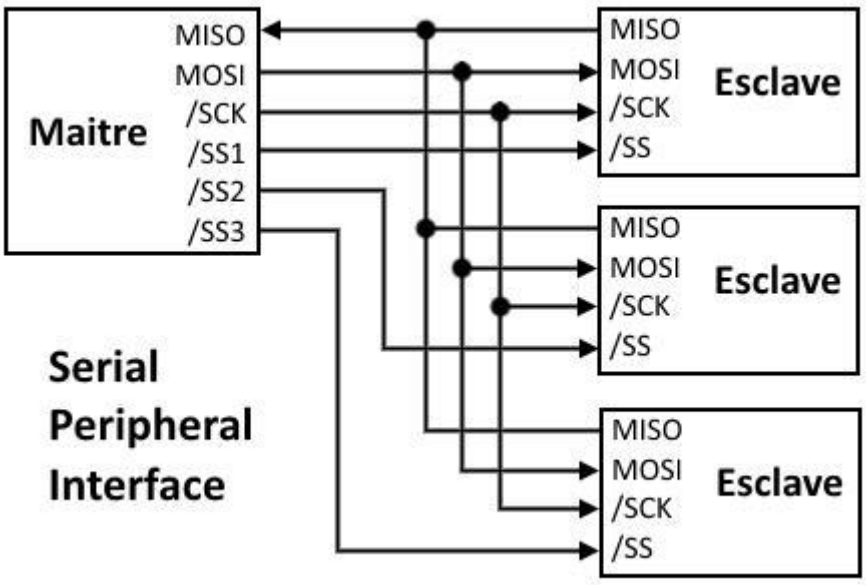


Diagramme câblage d'un bus SPI

Un bus SPI est composé de quatre fils : deux fils de données (`MOSI` et `MISO`), un fil d'horloge (`SCK`) et un fil d'adressage par périphérique (`/CS` ou `/EN`).

Les trois premiers fils sont communs à tous les périphériques d'un même bus SPI. Le quatrième fil (`/CS`) est unique par périphérique et permet au maître de choisir avec quel périphérique il souhaite communiquer.

Il est possible d'avoir plusieurs périphériques SPI sur un même bus (exemple : le module Ethernet et la carte SD de la shield Ethernet officielle). Le maître communique avec un et unique périphérique SPI à la fois en sélectionnant le périphérique de son choix via sa broche `/CS` ("Chip Select").

Par convention, mettre la broche `/CS` d'un périphérique SPI à `LOW` signifie qu'on souhaite communiquer avec celui-ci. Mettre la broche `/CS` à `HIGH` signifie que le périphérique doit libérer le bus et rester silencieux (en gros on lui dit de se taire, gentiment).

Lors du choix de la shield SD, il est **impératif** de connaitre à l'avance les broches que celle-ci va utiliser. Car en plus des broches `D11` à `D13` du port SPI, la shield SD va obligatoirement utiliser une quatrième broche (`/CS`) pour activer la communication avec la carte SD (cf parenthèse ci-dessus).

Si vous utilisez deux shields utilisant le port SPI, ce n'est pas un problème tant que les deux shields utilisent des broches d'activation différentes. Si par contre deux shields utilisent la (ou les) même(s) broche(s) d'activation, ou utilisent les broches du port SPI pour autre chose que le port SPI, cela ne fonctionnera pas. Pire, ce conflit de câblage peut entrainer des dommages physiques aux composants de la carte Arduino et des shields. Il est donc important de bien choisir ces shields.

N.B. Il arrive parfois que deux shields utilisent les mêmes broches. Par exemple, il arrive fréquemment que les shields utilisant le port SPI utilisent la broche `D4` ou `D10` pour `/CS`.

Certaines shields permettent de recâbler la broche en question sur une autre broche de son choix. Si vous êtes vraiment embêté, vous pouvez toujours déconnecter la broche problématique et souder un fil pour recâbler cette broche autre part. Cela demande un peu de doigté et un fer à souder, mais c'est tout à fait possible.

À noter que, qu'importe la shield que vous choisirez, si celle-ci utilise le port SPI, elle bloquera la broche `D10` (Arduino UNO et similaire) ou `D53` (Arduino Mega) en sortie. On verra pourquoi plus tard, mais gardez bien cela en tête quand vous ferez votre liste de course.

PS Les cartes Arduino Leonardo ne sont pas compatibles avec la plupart des shields SD du commerce, sauf indication contraire.

Choix de la carte SD

Diverses cartes SD

Le choix de la shield SD est important, mais le choix de la carte SD l'est tout autant.

Toutes les cartes SD ne sont pas compatibles avec les cartes Arduino. Cela est dû au fait que les cartes Arduino communiquent avec la carte SD via un bus SPI. La communication en mode SPI n'est pas la méthode classique de communication avec une carte SD. Ce mode un peu spécial est un héritage des toutes premières versions de cartes SD. Vous vous en doutez, depuis les premières cartes SD, il y a eu beaucoup de changement et les normes se sont enchainées au fil des années.

Les cartes SDXC ne sont PAS compatibles. De même que les cartes d'une capacité supérieure à 32Go.

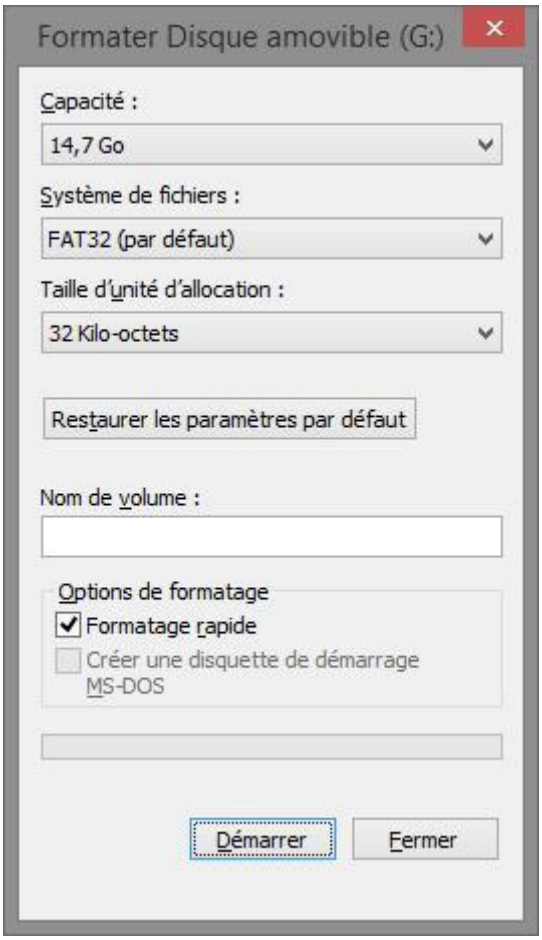
Les cartes SDHC (capacité supérieure à 4Go) sont compatibles la plupart du temps, qu'importe leur norme de vitesse (class 2, 4, 10, UI, etc.). Pour maximiser vos chances d'avoir une carte SD compatible, préférez des cartes de petites capacités (2Go, 4Go, ou 8Go) et avec une classe de vitesse "lente" (class 2, 4, 10).

Personnellement, je conserve précieusement mes cartes SD de petites capacités. Une carte SD de 256Mo peut sembler bien inutile de nos jours, mais pour une utilisation avec un montage Arduino, c'est le top.

Préparation de la carte SD

Avant de pouvoir être utilisé avec une carte Arduino, la carte SD doit être formatée. Cette étape est obligatoire.

Chaque système d'exploitation ayant son propre utilitaire de formatage, je vous laisse regarder sur internet la procédure de formatage d'une carte SD pour votre système. Pour les utilisateurs de Windows, je vous conseille d'utiliser l'utilitaire Rufus plutôt que celui fourni avec Windows.

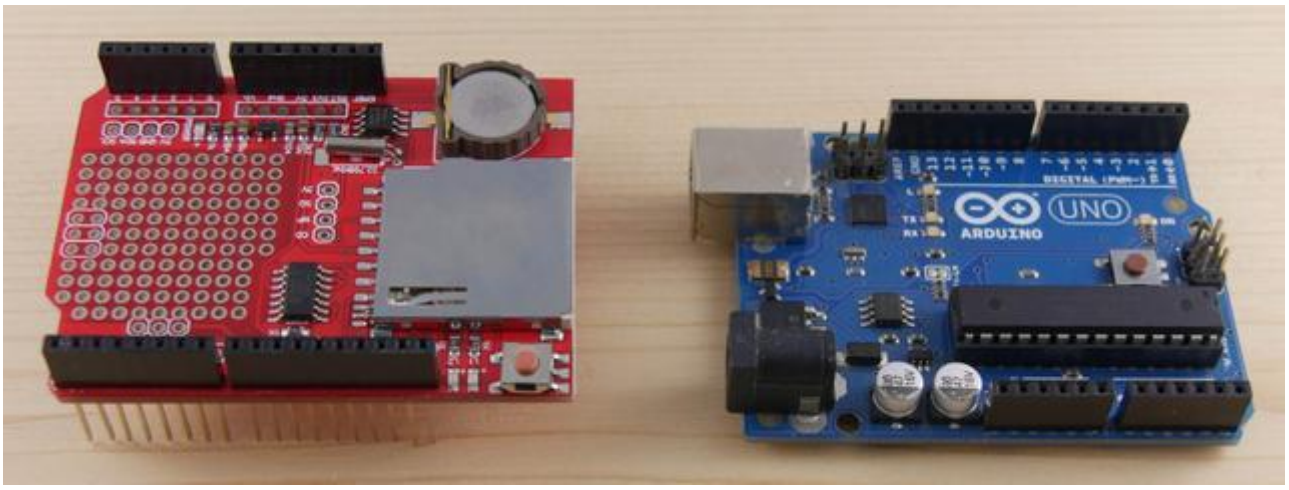


Utilitaire de formatage Windows

Le format de fichier doit être FAT16 (souvent abrégé FAT) ou FAT32. Pour les cartes de moins de 4Go, préférez FAT16 (plus léger), pour les autres cartes, utilisez FAT32. L'utilitaire de formatage vous demandera sûrement une taille de bloc ou d'allocation, laissez la valeur par défaut.

N.B. Le format exFAT n'est pas compatible avec les cartes Arduino. L'utilitaire Windows a tendance à forcer l'utilisation de ce format pour les cartes de plus de 8Go. Ce format ne fonctionnera pas, inutile de vous prendre la tête avec. Utilisez Rufus et demandez un "large FAT".

Montage de test



Shield SD chinoise et carte Arduino UNO

Pour ce tutoriel, j'utilise une carte Arduino UNO et une shield SD chinoise.

Ma carte utilise la broche `D10` pour la communication avec la carte SD. Pensez à modifier le numéro de broche dans `SD.begin()` en fonction de votre carte.

Initialisation de la carte SD

Pour communiquer avec la carte SD, nous allons utiliser la bibliothèque de code SD. Celle-ci est fournie de base avec le logiciel Arduino.

```
1 #include <SPI.h>
2 #include <SD.h>
```

La bibliothèque de code SD nécessite deux include : un pour le port SPI et un pour la communication avec la carte SD. Ceci devront être placés systématiquement en entête de programme.

```
1 bool SD.begin(broche_cs);
```

L'initialisation de la carte SD se fait via la fonction SD.begin().

Cette fonction prend en argument le numéro de la broche `/CS` qu'utilise votre shield SD. En retour, cette fonction retourne une valeur vraie (`true`) si l'initialisation a réussi ou une valeur fausse (`false`) en cas d'échec.

Il est impératif de gérer le cas d'erreur. Il est fréquent qu'une carte SD ne répondent pas ou ne puisse être lue (mauvais format de fichier par exemple). Si vous ne gérez pas le cas d'erreur, vous aurez des problèmes 😗

Autre subtilité à garder en tête : la broche `D10` (Arduino UNO et compatible) ou `D53` (Arduino Mega) doit impérativement être configurée en SORTIE. C'est une limitation matérielle. Même si vous n'utilisez pas cette broche, elle doit être en sortie pour que le bus SPI fonctionne.

Voici un exemple de code basique :

```
/**
 * Exemple de code de base pour initialiser une carte SD
 */

#include <SPI.h> // Pour la communication SPI
#include <SD.h>  // Pour la communication avec la carte SD


/* Broche CS de la carte SD */
const byte SDCARD_CS_PIN = 10; // TODO A remplacer suivant votre shield SD


/** Fonction setup() */
void setup() {

  /* Initialisation du port série (debug) */
  Serial.begin(115200);

  /* Initialisation du port SPI */
  pinMode(10, OUTPUT); // Arduino UNO
  //pinMode(53, OUTPUT); // Arduino Mega

  /* Initialisation de la carte SD */
  Serial.print(F("Init SD card... "));
  if (!SD.begin(SDCARD_CS_PIN)) {
    Serial.println(F("FAIL"));
    for(;;); //  appui sur bouton RESET
  }
  Serial.println(F("OK"));

  // TODO Votre code
}


/** Fonction loop() */
void loop() {

  // TODO Votre code
}
```

L'extrait de code ci-dessus est disponible en téléchargement sur cette page (le lien de téléchargement en .zip contient le projet Arduino prêt à l'emploi).

Ce code initialise le port série (pour debug), le port SPI et la carte SD. En cas d'erreur, le programme se bloque en ant l'appui sur le bouton `RESET` de la carte Arduino.

N.B. N'oubliez pas de modifier la valeur de la constante `SDCARD_CS_PIN` en fonction de votre shield SD. Pour les utilisateurs de carte Arduino Mega, pensez aussi à décommenter la ligne adéquate au niveau de l'initialisation du port SPI.

Opérations de haut niveau

La bibliothèque SD ne gère que les noms de fichiers ou dossiers au format DOS 8.3. Cela peut paraitre archaïque, mais c'est une limitation imposée à la fois à cause des brevets de Microsoft sur le format de fichier FAT et du peu de mémoire RAM des cartes Arduino. Dans les faits, on se contente très facilement d'un nom de fichier ou de dossier de huit caractères.

PS Les noms de fichiers sont insensibles à la case. `Toto.txt` et `TOTO.TXT` représentent le même fichier.

N.B. Je fais la distinction entre fichier et dossier dans les chapitres ci-dessous. Cependant, techniquement, un dossier EST un fichier (spécial). Vous ne pouvez bien sûr pas écrire des données dans un dossier, mais techniquement, un dossier est bel et bien un fichier. Ne soyez donc pas étonné de voir que le même type d'objet est utilisé pour manipuler un fichier et un dossier.

Vérifier la présence d'un fichier ou d'un dossier

```
1 bool SD.exists(chemin_fichier);
```

Pour vérifier la présence d'un fichier ou d'un dossier sur la carte SD, vous pouvez utilisez la fonction SD.exists().

Cette fonction prend en argument le nom du fichier ou du dossier. Si vous souhaitez vérifier la présence d'un fichier dans un sous dossier, il vous suffit de passer le chemin d'accès complet vers ce fichier, avec "/" comme séparateur. Cette fonction retourne un booléen vrai (`true`) si le fichier ou dossier existe, ou faux (`false`) si le fichier ou dossier n'existe pas.

Exemples :

```
1 SD.exists("toto.txt"); // Test si un fichier "toto.txt" existe à la racine de la carte SD
2 SD.exists("foobar"); // Test si un dossier "foobar" existe à la racine de la carte SD
3 SD.exists("foobar/toto.txt"); // Test si un fichier "toto.txt" existe dans le dossier "foobar"
```

Code d'exemple :

```
1 if(SD.exists("toto.txt")) {
```

```
  Serial.println(F("Fichier toto.txt OK"));
} else {
  Serial.println(F("Fichier toto.txt introuvable"));
}
```

Créer un dossier ou un sous-dossier

```
bool SD.mkdir(chemin_dossier);
```

Pour créer un dossier ou un chemin de dossiers, vous pouvez utiliser la fonction SD.mkdir().

Cette fonction prend en argument le nom du dossier à créer. Si vous souhaitez créer un sous-dossier (exemple "a/b/c"), il vous suffit de passer le chemin complet, avec "/" comme séparateur. La fonction créera automatiquement tous les dossiers intermédiaires si nécessaire. Cette fonction retourne un booléen vrai (`true`) si la création réussie, ou faux (`false`) si la création du dossier n'a pas pu être faite.

N.B. Si le dossier existe déjà, la fonction retourne une valeur vraie (`true`). Vous pouvez donc demander la création d'un dossier sans vérifier qu'il existe avant. Cela n'entrainera pas d'erreur.

N.B. Si cette fonction retourne une valeur fausse (`false`), il y a de fortes chances que la carte SD est était retirée. Gérez correctement ce cas d'erreur si vous ne voulez pas avoir de problèmes par la suite.

Exemples :

```
SD.mkdir("foobar"); // Créer un dossier "foobar" à la racine de la carte SD
SD.mkdir("foobar/toto"); // Créer un sous dossier "toto" dans le dossier "foobar" (créer aussi le dossier "foobar" s'il n'existe pas)
```

Supprimer un dossier ou un sous-dossier

```
bool SD.rmdir(chemin_dossier);
```

Pour supprimer un dossier, vous pouvez utiliser la fonction SD.rmdir().

Cette fonction prend en argument le nom du dossier à supprimer. Si vous souhaitez supprimer un sous-dossier, il vous suffit de passer le chemin complet, avec "/" comme séparateur. Cette fonction retourne un booléen vrai (`true`) si la suppression réussie, ou faux (`false`) si la suppression du dossier n'a pas pu être faite.

N.B. Le dossier doit être vide, sinon la suppression échouera. Ceci est une sécurité permettant d'éviter de supprimer par erreur un dossier contenant d'autres dossiers ou fichiers.

N.B. Si le dossier n'existe pas, la valeur de retour est non définie. En d'autres termes, vérifiez TOUJOURS que le dossier existe avant de le supprimer.

Exemples :

```
SD.rmdir("foobar"); // Supprime le dossier "foobar" à la racine de la carte SD
SD.rmdir("foobar/toto"); // Supprime le sous dossier "toto" dans le dossier "foobar"
```

Code d'exemple :

```
if(SD.exists("foobar")) {
  if(!SD.rmdir("foobar")) {
    Serial.println(F("Erreur suppression dossier"));
    for(;;); // Attend appui sur bouton RESET
  }
}
```

Même code, mais sous la forme d'une fonction réutilisable :

```
/**  Fonction de suppression d'un dossier avec gestion d'erreur */
void deleteDirectory(const char* filename) {
  if(SD.exists(filename)) {
    if(!SD.rmdir(filename)) {
      Serial.println(F("Erreur suppression dossier"));
      for(;;); // Attend appui sur bouton RESET
    }
  }
}

// Exemple d'utilisation
deleteDirectory("foobar");
```

Supprimer un fichier

```
bool SD.remove(chemin_fichier);
```

Pour supprimer un fichier, vous pouvez utiliser la fonction SD.remove().

Cette fonction prend en argument le nom du fichier à supprimer. Si vous souhaitez supprimer un fichier dans un sous dossiers, il vous suffit de passer le chemin complet, avec "/" comme séparateur. Cette fonction retourne un booléen vrai (`true`) si la suppression réussie, ou faux (`false`) si la suppression du fichier n'a pas pu être faite.

N.B. Si le fichier n'existe pas, la valeur de retour est non définie. En d'autres termes, vérifiez TOUJOURS que le fichier existe avant de le supprimer.

Exemples :

```
SD.remove("toto.txt"); // Supprime le fichier "toto.txt" à la racine de la carte SD
SD.remove("foobar/toto.txt"); // Supprime le fichier "toto.txt" dans le dossier "foobar"
```

Code d'exemple :

```
if(SD.exists("toto.txt")) {
  if(!SD.remove("toto.txt")) {
    Serial.println(F("Erreur suppression fichier"));
    for(;;); // Attend appui sur bouton RESET
  }
}
```

Même code, mais sous la forme d'une fonction réutilisable :

```
/**  Fonction de suppression d'un dossier avec gestion d'erreur */
void deleteFile(const char* filename) {
  if(SD.exists(filename)) {
    if(!SD.remove (filename)) {
      Serial.println(F("Erreur suppression fichier"));
      for(;;); // Attend appui sur bouton RESET
    }
  }
}

// Exemple d'utilisation
deleteFile("toto.txt");
```

Ouvrir un fichier pour lecture ou écriture

```
File SD.open(chemin_fichier);
File SD.open(chemin_fichier, mode);
```

Pour ouvrir un fichier, il convient d'utiliser la fonction SD.open().

Cette fonction permet d'ouvrir un fichier en lecture (défaut) ou en écriture. Plusieurs fichiers peuvent être ouverts simultanément depuis la version 1.0 du logiciel Arduino, dans la limite de la mémoire RAM disponible (voir chapitre bonus).

Cette fonction prend en argument le chemin complet vers le fichier à ouvrir, avec "/" comme séparateur de dossiers. Par défaut, le fichier est ouvert en lecture seule. Il est possible d'ouvrir ou de créer un fichier en l'ouvrant en écriture. Le choix du mode d'ouverture se fait via le second argument (optionnel). Deux constantes sont acceptées : `FILE_READ` pour une ouverture en lecture seule et `FILE_WRITE` pour une ouverture en écriture.

N.B. Si un fichier est ouvert en écriture, mais n'existe pas sur la carte SD, il est automatiquement créé.

N.B. Lorsqu'un fichier est ouvert en lecture, la lecture commence au début du fichier. Lorsqu'un fichier est ouvert en écriture, l'écriture commence en fin du fichier (ajout de données).

Si l'ouverture du fichier réussit, vous obtiendrez en retour un objet `File`. Cet objet vous permettra de manipuler le contenu du fichier via les fonctions ci-dessous. Cependant, si

l'ouverture du fichier échoue, vous obtiendrez quand même un objet `File`, mais celui-ci ne sera pas utilisable.

Pour déterminer si le fichier a été ouvert correctement ou non, il suffit de faire un test avec un `if` :

```
File fichier = SD.open("toto.txt");
if(!fichier) {
  // Erreur d'ouverture du fichier
  Serial.println(F("Impossible d'ouvrir le fichier"));
  for(;;); // Attend appui sur bouton RESET
}

// Votre code
```

N.B. Vérifiez systématiquement si l'ouverture du fichier a réussi ou non. Si vous ne prenez pas le temps de faire cette vérification, vous allez avoir de gros problèmes par la suite avec votre code.

Opérations possibles sur un fichier

Nous allons à présent nous intéresser aux opérations possibles sur un fichier (un vrai, pas un dossier).

Obtenir la taille en octets du fichier

```
unsigned long File.size();
```

On peut tout d'abord obtenir la taille totale du fichier en octets via la fonction File.size().

La valeur retournée est un entier positif (`unsigned long`) équivalent à la taille actuelle du fichier en octets.

Exemple d'utilisation :

```
File fichier = SD.open("toto.txt");
// Gestion erreur ici

Serial.print(F("Taille toto.txt : "));
Serial.print(fichier.size());
Serial.println(F(" octets"));
```

Compter le nombre d'octets restants à lire

Lorsque l'on lit un fichier, il est important de savoir combien d'octets de données restent à lire.

```
int File.available();
```

La fonction File.available() permet de savoir combien d'octets sont disponibles en lecture dans le fichier. La valeur de retour est un nombre entier.

Si cette fonction retourne 10 par exemple, cela signifie que vous pouvez encore lire 10 octets avant d'arriver à la fin du fichier.

Exemple d'utilisation :

```
File fichier = SD.open("toto.txt");
// Gestion erreur ici

while (fichier.available() > 0) {
  // Affiche un octet du fichier tant qu'il y a des données à lire
  Serial.write(fichier.read());
}
```

Lire un certain nombre d'octets

```
int File.read();
int File.peek();
```

Pour lire un octet depuis le fichier, vous avez deux solutions : utiliser la fonction File.read() qui lit un octet dans le fichier puis passe au suivant ou utiliser la fonction File.peek() qui lit un octet dans le fichier sans pour autant passer au suivant.

Si vous voulez lire normalement un octet dans le fichier, utilisez `File.read()`. Si vous voulez voir ce que retournera le prochain appel à la fonction `File.read()`, utilisez `File.peek()`. La fonction `File.peek()` est assez peu souvent utilisée, sauf dans des cas un peu particuliers où il est nécessaire d'avoir un aperçu du prochain octet sans passer au suivant.

Dans les deux cas, vous obtiendrez un `int` en retour de la fonction. Ces fonctions ne retournent pas directement un `byte` de manière à pouvoir retourne `-1` en cas d'erreur. Dans les faits, vous pouvez stocker directement le retour de la fonction dans un `byte` sans problème.

Ces fonctions ne retournent `-1` que s'il n'y a plus de données à lire, mais comme vous avez testé le nombre d'octets restant avant d'appeler `File.read()` vous ne devriez pas avoir de soucis. Vous n'allez pas lire des données sans vérifier avant qu'il reste quelque chose à lire, hein, rassurez-moi 

Exemple d'utilisation :

```
File fichier = SD.open("toto.txt");
// Gestion erreur ici

while (fichier.available() > 0) {
  // Affiche un octet du fichier tant qu'il y a des données à lire
  Serial.write(fichier.read());
}
```

Pour lire plusieurs octets d'un seul bloc, vous pouvez utiliser la version spécialisée de `File.read()` :

```
int File.read(byte* buffer, int taille);
```

Cette version prend en argument un tableau de `byte` et la taille du tableau en question. La fonction retourne le nombre d'octets lus et copiés dans le tableau de `byte`.

Exemple d'utilisation :

```
const int TAILLE_BUFFER = 8;

File fichier = SD.open("toto.txt");
// Gestion erreur ici

while (fichier.available() >= TAILLE_BUFFER) {
  byte buffer[TAILLE_BUFFER];
  int nbytes = fichier.read(buffer, TAILLE_BUFFER);
  if (nbytes !=  TAILLE_BUFFER) {
    // Erreur de lecture
  }
  // Votre code
}
```

Écrire un certain nombre d'octets

```
int File.write(byte octet);
int File.write(byte* buffer, int taille);
```

Pour écrire des données dans un fichier ouvert en écriture, vous pouvez utiliser la fonction File.write().

La fonction `File.write()` est disponible en deux versions : une version permettant d'écrire un octet à la fois et une version permettant d'écrire un bloc d'octets.

Dans les deux cas, cette fonction retourne le nombre d'octets écrits de manière à détecter une erreur d'écriture.

Exemple d'utilisation :

```
File fichier = SD.open("toto.txt", FILE_WRITE);
// Gestion erreur ici

while (Serial.available()) {
  fichier.write(Serial.read());
}
```

Écrire du texte, des chiffres, etc.

Écrire des octets, c'est bien beau, mais ce n'est pas très pratique. Si vous souhaitez écrire du texte ou des nombres dans le fichier, vous pouvez utiliser les fonctions `File.print()` et `File.println()`.

Ces deux fonctions se comportent exactement comme `Serial.print()` et `Serial.println()`.

Exemple d'utilisation :

```
File fichier = SD.open("toto.txt", FILE_WRITE);
// Gestion erreur ici

fichier.println(F("Bonjour le monde"));
```

Obtenir et / ou modifier la position actuelle dans le fichier

Il est parfois nécessaire de choisir l'endroit où l'on souhaite écrire ou lire des données.

Pour obtenir la position actuelle dans le fichier, il convient d'utiliser la fonction File.position(). La valeur de retour est un nombre entier long positif (`unsigned long`).

Pour revenir quelque part dans le fichier, il convient d'utiliser la fonction File.seek(). Cette fonction prend en argument la nouvelle position dans le fichier (`unsigned long`) et retourne un booléen vrai (`true`) en cas de succès ou un booléen faux (`false`) en cas d'erreur.

Exemple d'utilisation :

```
File fichier = SD.open("toto.txt", FILE_WRITE);
// Gestion erreur ici

Serial.println(fichier.position()); // Affiche 0
fichier.println(F("Bonjour le monde"));
Serial.println(fichier.position()); // Affiche 18

fichier.seek(0);
fichier.println(F("Bonjour les makers"));
// Le fichier contient désormais "Bonjour les makers" au lieu de "Bonjour le monde"
```

Forcer l'écriture immédiate des données en attente

Faite le test suivant : ouvrez un fichier en écriture, écrivez un morceau de texte et retirer la carte SD. Que contient le fichier ? Rien ? C'est normal.

Pour éviter de tuer la carte SD en écrivant octet par octet dans sa mémoire, la bibliothèque SD garde en mémoire un tableau d'octets à écrire. L'écriture physique n'est réalisée que quand ce tableau est plein, que l'on ferme le fichier ou que l'on appelle la fonction File.flush().

La fonction `File.flush()` force l'écriture immédiate des données en attente. Cette fonction n'est normalement pas nécessaire, car l'écriture des données se fait à la fermeture du fichier. Cependant, dans certaines applications comme des relevés de mesures, on peut avoir besoin de retirer la carte SD en cours d'exécution. En appelant `File.flush()` après chaque écriture d'une mesure, on s'assure de ne pas perdre de données.

Exemple d'utilisation :

```
File fichier = SD.open("toto.txt", FILE_WRITE);
// Gestion erreur ici

fichier.println(F("Bonjour le monde"));
fichier.flush(); // Les données sont désormais écrites sur la carte SD
```

Attention : si vous appelez `File.flush()` trop souvent, vous allez réduire considérablement la durée de vie de votre carte SD. Si vous devez appeler `File.flush()`, faite en sorte que ce soit le moins souvent possible.

Fermer le fichier

Vous avez fini de lire ou d'écrire dans un fichier. Vous n'en avez plus besoin. Vous allez donc appeler la fonction File.close() pour faire le ménage et libérer de la mémoire RAM.

Vous devez TOUJOURS appeler la fonction `File.close()` après avoir fini de travailler avec un fichier. Cela permet de finaliser l'écriture des données, de libérer la mémoire RAM et donc d'éviter des pertes de données ou des plantages aléatoires.

Exemple d'utilisation :

```
File fichier = SD.open("toto.txt", FILE_WRITE);
// Gestion erreur ici

fichier.println(F("Bonjour le monde"));
fichier.close();
```

N.B. Si le fichier n'est pas ouvert (erreur d'ouverture par exemple), inutile d'appeler la fonction `File.close()` 

Opérations possibles sur un dossier

N.B. Bien qu'un dossier soit un fichier spécial, vous ne pouvez pas écrire de données dans un dossier.

Vérifier si le dossier est bien un dossier

Question : comment savoir si le fichier que je viens d'ouvrir est un fichier ou un dossier ?

Réponse : facile, avec la fonction File.isDirectory().

La fonction `File.isDirectory()` retourne un booléen vrai (`true`) si le fichier courant est un dossier, ou un booléen faux (`false`) si c'est un simple fichier.

Exemple d'utilisation :

```
File fichier = SD.open("toto.txt");
// Gestion erreur ici

if (fichier.isDirectory()) {
  Serial.println(F("C'est un dossier"));
} else {
  Serial.println(F("C'est un fichier"));
}
```

Ouvrir le fichier suivant dans le dossier

Dans un dossier, il y a un certain nombre de fichiers.

La fonction File.openNextFile() permet d'ouvrir le fichier suivant dans un dossier.

N.B. L'ordre d'ouverture des fichiers est lié à l'ordre de création des fichiers sur la carte SD. Il n'y a pas de possibilité de tri par date, nom ou autre.

Exemple d'utilisation (liste tous les fichiers à la racine de la carte SD) :

```
File dossier = SD.open("/");
File fichier = dossier.openNextFile();
while (fichier) {
  Serial.println(fichier.name());
  fichier.close();
  fichier = dossier.openNextFile();
}
```

Revenir au premier fichier du dossier

Si vous souhaitez revenir au premier fichier d'un dossier de manière à recommencer une boucle avec `File.openNextFile()`, vous pouvez appeler la fonction File.rewindDirectory().

Exemple d'utilisation :

```
File dossier = SD.open("/");

File fichier_1 = dossier.openNextFile();
// ...

dossier.rewindDirectory()

File fichier_2 = dossier.openNextFile();
// fichier_1 est fichier_2 pointe vers le même fichier
```

Fermer le dossier

Comme quand vous avez fini de lire ou d'écrire dans un fichier. Vous devez appeler la fonction File.close() pour faire le ménage et libérer la mémoire RAM.

Si vous utilisez la fonction `File.openNextFile()`, pensez à bien fermer les fichiers temporaires avant de boucler. Si vous laissez les fichiers temporaires ouverts, vous finirez par manquer de mémoire RAM et vous vous retrouverez avec des bugs complètement délirants de manière aléatoire.

Bonus : Lister tous les fichiers dans un dossier

Il est possible d'obtenir le nom d'un fichier (ou dossier) ouvert au moyen de la fonction `File.name()`.

On peut donc lister récursivement tous les fichiers sur la carte SD si on le souhaite :

```
1  void listDirectoryFiles(File directory, int indent) {
2
3    /* Ouvre le premier fichier */
4    File entry = directory.openNextFile();
5    while (entry) {
6
7      /* Affiche l'indentation */
8      for (byte i = 0; i < indent; i++) {
9        Serial.write(' ');
10     }
11
12     /* Affiche le nom du fichier courant */
13     Serial.print(entry.name());
14
15     /* Affichage récursif pour les dossiers */
16     if (entry.isDirectory()) {
17       Serial.println("/");
18       listDirectoryFiles(entry, indent + 1);
19
20     } else {
21
22       /* Affiche la taille pour les fichiers */
23       Serial.println(entry.size(), DEC);
24     }
25
26     /* Ferme le fichier et ouvre le suivant */
27     entry.close();
28     entry = directory.openNextFile();
29   }
30
31   /* Revient au début du dossier */
32   directory.rewindDirectory();
```

```
33 }
34
35 // Exemple d'utilisation
36 File dossier = SD.open("/");
37 listDirectoryFiles(dossier, 0);
```

Bonus : Surveiller la mémoire RAM disponible

Si vous compilez le code de base du premier chapitre, vous obtiendrez le message suivant :

Le croquis utilise 6 556 octets (20%) de l'espace de stockage de programmes. Le maximum est de 32 256 octets. Les variables globales utilisent 795 octets (38%) de mémoire dynamique, ce qui laisse 1 253 octets pour les variables locales. Le maximum est de 2 048 octets.

Sans même avoir ouvert le moindre fichier, la bibliothèque SD utilise presque la moitié de la mémoire RAM disponible. Ce n'est pas un bug. C'est normal.

Communiquer avec une carte SD et manipuler un système de fichier n'est pas une opération triviale. Cela demande beaucoup de ressources, surtout en mémoire RAM. Il est donc important de faire attention à son utilisation de la mémoire RAM.

La fonction ci-dessous permet de calculer le nombre d'octets de mémoire RAM disponible :

```
1 /** Utilitaire pour calculer la mémoire RAM disponible */
2 int freeRam() {
3   extern int __heap_start, *__brkval;
4   int v;
5   return (int) &v - (__brkval == 0 ? (int) &__heap_start : (int)
__brkval);
6 }
```

Placez quelques `Serial.println(freeRam());` dans votre code à des endroits clefs durant le debug pour avoir une idée de la mémoire RAM restante.

Si vous voyez la mémoire RAM disponible descendre en dessous des 128 octets, c'est le signe qu'il va falloir faire attention.

Articles pouvant vous intéresser

- Fabriquer un système d'acquisition de mesures (datalogger) avec une carte Arduino / Genuino