# Arduino´s SD-Library

09.09.2013, A.Schultze, DK4AQ

Quelle:
file:///C:/Programme/Arduino/Arduino1.0.5/reference/SD.html

Übersicht der Funktionen der SD-Library

Die SD-Library arbeitet an dieser Stelle mit Objekten. Es werden 2 Klassen vzur Verfügung gestellt :

**Klasse SD:**
**Verwaltung der SD-Karte mit Filenamen und Verzeichnis-struklturen**

- **begin()**
- **exists()**
- **mkdir()**
- **open()**
- **remove()**
- **rmdir()**

**Klasse Files:**
**Zugriff auf ein File**

- **available()**
- **close()**
- **flush()**
- **peek()**
- **position()**
- **print()**
- **println()**
- **seek()**
- **size()**
- **read()**
- **write()**
- **isDirectory()**
- **openNextFile()**
- **RewindDirectory()**
- **name()**

Andere libraries :
https://code.google.com/p/sdfatlib/downloads/list

# Objektorientierte Ausdrücke in  Arduino

Klassen sind „Schablonen“ für Objekte. Objekte sind konkret ablauffähige Programme, die einerseits Datenstrukturen enthalten („Atribute“) und andererseits Zugriffs-Funktionen („Methoden“).. Es können mehrere Objekte („Instanzen von Klassen“)der gleichen Klasse definiert („instantiiert“) werden. Die Wirkung zur Laufzeit ist, dass es soviele Variablensätze wie Objekte der gleichen Klasse gibt, der Code jedoch nur einmal existiert und der nun zugehörige Variablensätze bei Aufruf einer Methode abarbeitet. Die Inhalte der Variablensätze bleiben auch nach Durchlauf erhalten, sind jedoch gekapselt.

Ob Objektorientierte Programmierung  (OOP) in Steuerungsprojekten Sinn machen, darüber kann man sich streiten. Wir wollen auch Libraries nutzen die auf diese Weise geschrieben wurden.

**Beispiel:**

```
File MeinFile      //Instantiierung der Klasse File in das Objekt MeinFile
```

Damit hat MeinFile die richtige Variablenstruktur und ausserdem alle Funktionen von File verfügbar.

*MeinFile.readf, MeinFile.write, MeinFile.println* sind z.B. gültige Aufrufe
Uanabhängig davon können andere Objekte der fleichen Klasse bearbeitet werden.

# SD-Klasse

## SD.begin()

Die Funktion initialisiert die SD-Library und initialisiert den SPI-Bus am Kartenanschluss. Die (Micro-)SD-Karte wird dabei über die SPI-Pins des jeweiligen Boards angeschlossen. Auch ein CS/SS is zwingend notwendig. Es muss ein Pin als Ausgang dafür definiert werden, z.B. der Pin 10.

*SD.begin(cspin)*

**Übergabeparameter:**

cspin kann optional übergeben werden. Cspin beschreibt die Nummer des Pins (z.B. D6), der an den Chip Select (CS oder SS) der SD-Karte angeschlossen ist. Dieser Pin ist der CS. der im SPI-Protokoll verwendet wird.

**Rückgabewert:**

- true wenn Karte vorhanden ist
- false wenn keine Karte vorhanden ist

# SD.exists()

Mit dieser Funktion kann man testen, ob es eine bestimmtes Verzeichnis oder ein bestimmtes Datenfile auf der Karte gibt.

**Übergabeparameter:**

*SD.exists(filename)*

*filename* ist der Name des Files in Anführungszeichen oben (z.B. “FILE.TXT“) oder ein Verzeichnis in Querstrichen (z.B. /MESSUNGEN/)

**Rückgabewert:**

- *true* wenn File oder Verzeichnis existiert
- *false* wenn File oder Verzeichnis existiert.

# SD.Mkdir()

Erzeugt ein Verzeichnis auf der SD-Karte.
Create a directory on the SD card. This will also create any intermediate directories that don’t already exists; e.g. SD.mkdir("a/b/c") will create a, b, and c.

**Übergabeparameter:**

SD.mkdir(dirname)

*dirname* ist der Name des anzulegenden Verzeichnisses in Anführungszeichen oben (z.B. “DIR3“. Es können auch Verzeichnisbäume angelegt werden. Z.B. SD.mkdir("a/b/c") wird die verzeichnisse a, b, und c als Baum mit Unterverzeichnissen anlegen.

**Rückgabewert:**

- true wenn Verzeichnis erfolgreich angelegt wurde
- false wenn Verzeichnis NICHTangelegt werden konnte

# SD.open()

Die Funktion  öffnet ein File. Wenn es zum Schreiben geöffnet wird, dann wird das File erst angelegt, falls es noch nicht existiert.

**Überfgabeparameter:**

*SD.open(filepath)*
*SD.open(filepath, mode)*

*filenpath* ist der Name des Files inklusive des Zugriffspfades über die Verzeichnisse. Die Verzeichnisebenen werde durch einen Schrägstrich rechts getrennt. Der Parameter ist vom Typ character-Zeiger ( char* ). Es kann auch ein String in Anführungszeichen oben “ eingegeben werden.

*mode* ist optional der zweck zu dem das File geöffnet wird.:

- *FILE_READ*: öffnen zum Lesen, die Operation beginnt am Anfang des Files
- *FILE_WRITE*: öffnen zum Lesen und Schreiben, die Operation beginnt am Ende des Files
Defaultmässig wird immer FILE_READ genutzt.

**Rückgabewert:**

Qbjekt des Typs *File* :
- wenn File erfolgreich geöffnet wurde, Zeiger auf die Datenstrukturen des Objekt
- wenn File nicht geöffnet werden konnte, dann wird 0 oder false zurückgeliefert

# SD.remove()

Löscht ein File von der SD-Karte

*SD.remove(filename)*

**Übergabeparameter:**

*filename* ist der Name des Files was gelöscht werden soll. Es können verzeichnispfade enthalten sein, die durch Schrägstrich rechts (/) getrennt werden.

**Rückgabewert:**

- true: Der Löschvorgang ist gelungen
- false: Der Loschvorgang ist nicht gelungen (z.B. wenn das File nicht existierte)

# SD.rmdir()

Entfernt ein Verzeichnis von der SD-Karte. Das Verzeichnis muss leer sein.

*SD.rmdir(dirname)*

**Übergabeparameter**

Dirname ist der Name des Verzeichnisses. Es können Unterverzeichnisse angegeben werden, die durch Schrägstrich rechts (/) getrennt werden.

**Rückgabewert:**

- true: Löschen des Verzeichniosses ist gelungen.
- false: Löschen ist nicht gelungen

# File-Klasse

## file.available()

Prüft, ob es Bytes im  File gibt, die gelesen werden können.
( *available()* ist abgeleitet aus der *Stream utility* Klasse.)

*file.available()*

**Übergabeparameter:**

*file* ist ein Objekt vom Typ File. Es wurde von SD.open() zurückgeliefert (referenziert).

**Rückgabewert:**

Anzahl der Bytes, die noch verfügbar sind (Typ int)

# file.flush()

Sorgt dafür, daß alle geschriebenen Bytes physikalisch auf die SD-Karte geschrieben werden. Normalerweise wird das beim Schliessen des Files mit file.close() automatisch durchgeführt.
( *file.flush()* ist abgeleitet von der *Stream utility* Klasse.)

*file.flush()*

**Übergabeparameter:**

*file* ist ein Objekt der File-Klasse im Mikrorechner. File wird von *SD.open()* zurückgeliefert (referenziert).

**Rückgabewert:**

Keine

# file.peek()

Es wird ein Byte aus dem Objekt file gelesen ohne den internen Byte-Zeiger zu verändern.
Beim nächsten file.read() wird genau das gleiche Byte gelesen.
( *peek()* ist eine Ableitung der *Stream utility* Klasse.)

*file.peek()*

**Übergabeparameter:**

File ist ein Objekt der Klasse File. Es wird von file.open() zurückgeliefert (referenziert)

**Rückgabewerte:**

Das nächste Byte oder -1 wenn keines mehr verfügbar ist.

# file.position()

Liefert den Stand des internen Byte-Zeigers im File ( D.h. Die Adresse des nächsten Bytes, welches durch das nächste file.read() gelesen wird.

*file.position()*

**Übergabeparameters:**

File ist ein Objekt vom Type File im Mikrorechner. Es wird von SD.open() zurückgeliefert (referenziert).
:
**Rückgabewert:**

Die position des internen Byte-Zeigers  (unsigned long)

# file.close()

Schliesst das File und sorgt dafür, das alle Daten des File-Objektes im Mikrorechner physikalisch auf die SD-Karte gespeichert wird.

*file.close()*

**Übergabeparameter:**

file ist das Objekt vom Typ File im Mikrorechner,. Es wurde von SD.open zurückgeliefert (referenziert).

**Rückgabewert:**

keiner

# file.print()

Es werden Daten in das File ´gedruckt´ Es werden Zahlen als ein Reihenfolge von Ziffern als ASCII-Character abgelegt. Z.B. wird der Wert 123 als drei Zeichen ’1’, ’2’, ’3’ abgelegt.

*file.print(data)*
*file.print(data, BASE)*

**Übergabeparameter:**

*file* ist ein Objekt des Typs File. Es wird von *SD.open()* zurückgeliefert (referenziert).
*data* ist ein einzelnes Zeichen, welches abgelegt werden soll.
*BASE* (optional) ist die Darstellung des Zeichens: BIN binär (base 2), DEC dezimal (base 10), OCT oktal (base 8), HEX hexadezimal (base 16).

**Rückgabewert:**

file.print() liefert die Anzahl der geschriebenen Bytes (Typ byte)

# file.println()

Druckt Daten in das File, gefolgt von “carriage return“ und “newline“. Es werden Zahlen als ein Reihenfolge von Ziffern als ASCII-Character abgelegt. Z.B. wird der Wert 123 als drei Zeichen ’1’, ’2’, ’3’ abgelegt.

*file.println()*
*file.println(data)*
*file.print(data, BASE)*

**Übergabeparameter:**

*file* ist ein Objekt des Typs File. Es wird von *SD.open()* zurückgeliefert (referenziert).
*data* ist ein einzelnes Zeichen, welches abgelegt werden soll.
*BASE* (optional) ist die Darstellung des Zeichens: BIN binär (base 2), DEC dezimal (base 10), OCT oktal (base 8), HEX hexadezimal (base 16).

**Rückgabewert:**

file.print() liefert die Anzahl der geschriebenen Bytes (Typ byte)

# file.seek()

Sucht die neue Position des Byte-Zeigers im File, auf die als nächstes geschrieben wird.
Der Wert muss zwischen 0 und der Filelänge liegen.

*file.seek(pos)*

**Übergangsparameters:**

*file* ist ein Objekt des Typs File. Es wird von *SD.open()* zurückgeliefert (referenziert).
Pos ist die Position  nach der gesucht wird (Typ unsigned long)

**Rückgabewert:**

- true: Position gefunden
- false: Position nicht gefunden
 (Typ boolean)

## file.size()

Liefert die Größe eines Files

*file.size()*

**Übergangsparameter:**

*file* ist ein Objekt des Typs File. Es wird von *SD.open()* zurückgeliefert (referenziert)

**Rückgabewert:**

Größe des Files in Bytes (Typ unsigned long)

## file.read()

Liest ein Byte vom File.
( read() iist abgeleitet aus der Stream utility Klasse.)

*file.read()*

**Bergabeparameter:**

*file* ist ein Objekt des Typs File. Es wird von *SD.open()* zurückgeliefert (referenziert)

**Rückgabewert:**

Das nächste Byte oder Character oder -1 wenn kein Byte mehr verfügbar ist.

## file.write()

Schreibt Date in das File

*file.write(data)*
*file.write(buf, len)*

**Übergabeparameter:**

*file* ist ein Objekt des Typs File. Es wird von *SD.open()* zurückgeliefert (referenziert)
*data* sind byte, char, oderr string (char *) welche(s) geschrieben werden soll(en).
*buf* ist ein Array von Werten aus dem geschrieben werden soll
*len* ist die Anzahl von elementen in *buf*

**Rückgasbewert:**

Anzahl der Bytes, die geschrieben wurden. (Typ byte)

# file.isDirectory()

Verzeichnisse werden formal wie Files behandelt. Die Funktion fragt ab, ob das Objekt ein Verzeichnisname ist.

*file.isDirectory()*

**Übergabepaparameter:**

*file* ist ein Objekt des Typs File. Es wird von *SD.open()* zurückgeliefert (referenziert). Auch Verzechnisse sind Objekte vom Typ File.

**Rückgabewert:**

- true: Objekt ist ein Verzeichnisname
- false: Objekt ist kein Verzeichnisname

(Typ boolean)

## file.openNextFile()

Erkennt das nächste File in einem Verzeichnis.

*file.openNextFile()*

**Übergabeparameter:**

*file* ist ein Objekt des Typs File. In dieser Funktion ist File ein Verzeichnis-Objekt. Es wird in einem Verzeichnis nach dem nächsten Objekt gesucht (File oder Verzeichnis)

**Rückgabewert:**

Referenz auf ein Objekt vom Typ File. Dieses Objekt ist das nächste File oder Verzeichnis. Mit *file.name()* kann der Name ermittelt werde. Mit *file.isDirectory()* kann festgestellt werden, ob es sich um ein Verzeichnis handelt

# file.rewindDirectory()

Die Funktion stellt die Untersuchung eines Verzeichnisses zurück auf den ersten Eintrag in einem Verzeichnis. Wird verwendet im Zusammenhang mit openNextFile()

*file.rewindDirectory()*

**Übergabeparameter:**

*file* ist ein Objekt des Typs File. In dieser Funktion ist File ein Verzeichnis-Objekt. Es handelt sich um das Verzeichnis, das auf den ersten Eintrag zurückgestellt werden soll.

**Rückgabewert:**

keiner

## file.name()

Ermittelt den Namen eines File-Objektes (File oder Verzeichnis).

*string = file.name()*

**Achtung !** Direkte Zuweisung ist bei Strings im Programmablauf nicht möglich. Wird in Verbindung mit *Serial.print()* oder *strcopy()* verwendet.

**Übergabeparameter:**

*file* ist ein Objekt des Typs File. Dies kann eine Verzeichnis- oder File-Objekt sein.

**Rückgabewert:**

Name des Objektes als String mit ASCII-Charactern