Arduino Ethernet+SD

December 19, 2011 13:28

Using the Arduino to browse files on an SD card remotely

Whatsit?

We just got the latest version of the Arduino Ethernet shield with a MicroSD card slot [http://www.adafruit.com/index.php?main_page=product_info&cPath=17_21&products_id=201] **and** I promised Bill Greiman I'd try out the latest version of his SdFatLib library [http://code.google.com/p/sdfatlib/downloads/list] so I decided to code up a simple Webified file browser. Its a quicky project and demonstrates what you can do, but it isn't 100% perfect so you should be ready to modify it if you'd like to do other stuff, 'K?

This is a good beginning to a logging web-monitor, or remote storage system.

Get familiar

Before you do this tutorial you'll want to get familiar with what you're working on. This **isn't** a beginner tutorial, its best used by those who already have quite a bit Arduino or microcontroller experience and are 'fluent' in C/C++/Java!

Also, read up on how to use the Ethernet shield [http://www.arduino.cc/en/Main/ArduinoEthernetShield] and Ethernet library [http://www.arduino.cc/en/Reference/Ethernet] . Then review my notes on SD card usage and installing the library we'll be using [http://www.ladyada.net/learn/../make/logshield/sd.html]

You should have already gotten the Ethernet shield working with your network setup, too.

While we're happy to share this code, please note that this project is just example code. **It is completely unsupported**. Enjoy it! Mod it! Hack it! But please don't expect more than what is posted here! You can download the latest code from GitHub (click Download Source in the top right) [http://github.com/adafruit/SDWebBrowse]

Initializing Micro-SD card on an Ethernet shield

The latest Arduino Ethernet shield [http://www.adafruit.com/index.php?main_page=product_info&cPath=17_21&products_id=201] comes with a handy MicroSD card slot so that you can store and retrieve data through the shield. Very handy! Lets show how to talk to the card.

> Be sure to have the very latest version of SdFatLib [http://code.google.com/p/sdfatlib/downloads/list] , as you'll need some of the newer capabilities!

First thing to note is that the **SS** (Slave Select) pin for the card is **digital 4** (although as of the writing of this mini-tutorial, the schematic [http://arduino.cc/en/uploads/Main/arduino-ethernet-shield-schematic.pdf] hasn't been updated, you'll have to trust me!)

Open up the **SdFatInfo** example sketch and change the line in **loop()** from

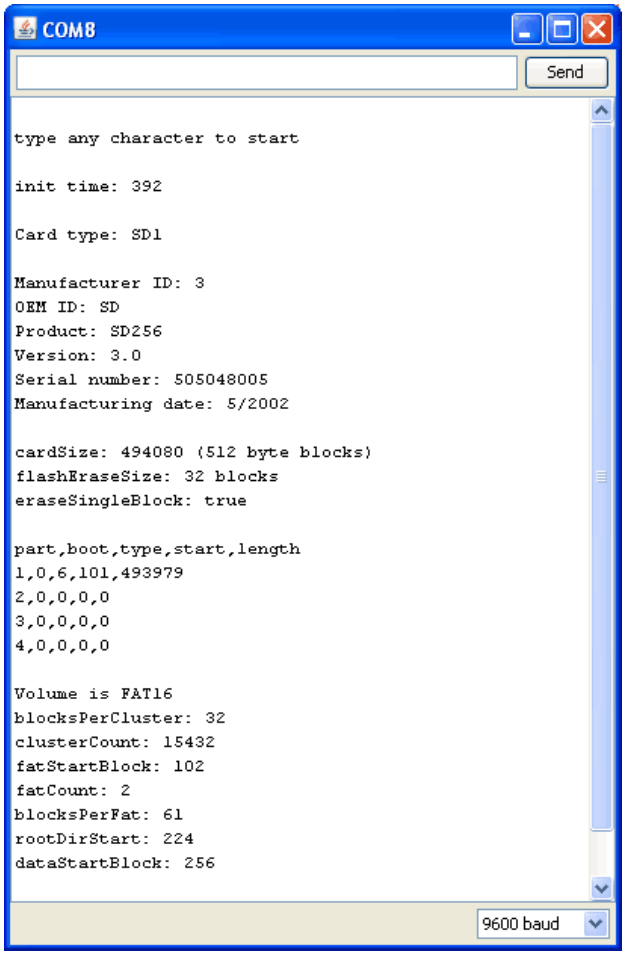
```
uint8_t r = card.init(SPI_HALF_SPEED);
```

To:

```
pinMode(10, OUTPUT);                       // set the SS pin as an output (necessary!)
digitalWrite(10, HIGH);                    // but turn off the W5100 chip!
uint8_t r = card.init(SPI_HALF_SPEED, 4);  // Use digital 4 as the SD SS line
```

Be sure to add those two extra lines right before-hand! They Enable the SPI interface. If you're on a Mega, use pin **53** instead of **10**

Now upload and test the card, you should see something like this:

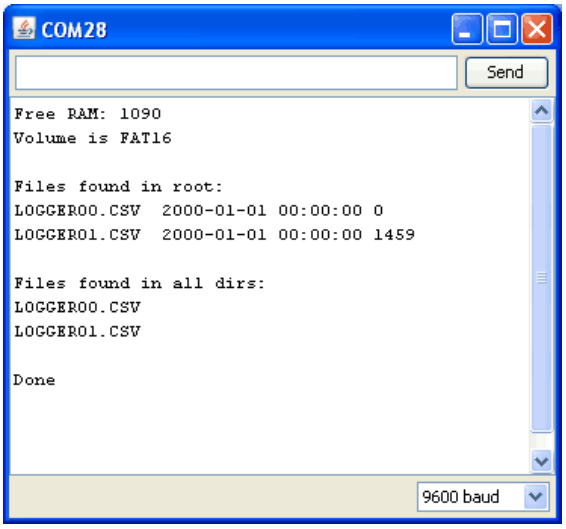


Indicating you talked to the card all right

List files

Put some text files on your SD card, using a computer, so that you have data to read. Make sure they are in the root directory, and not in a folder

Then run the **SdFatLs** example sketch from the SdFat library, you should see it list all the files you have on the card, again, you'll have to make the changes from above to update the **card.init()** part to the new **SS** pin



For example, the card I'll be using has two files on it from some previous datalogging.

Merge WebServer + SdFatLs

We'll begin by combining WebServer (the example sketch that comes with the **Ethernet** lib) and **SdFatLs** to make a web server that lists the files on the SD card. You can download the file here [http://github.com/adafruit/SDWebBrowse/blob/46a7d6a28f8b44d1a00bfddec987cf71d24da79a/SDWebBrowse.pde] (you'll need to copy&paste it, do so carefully!)then follow along!

Part one is the Ethernet and SD card objects and a simple error function (**void error_P(const char* str)**) that prints out errors and halts the program if there are serious problems.

You should, of cousre, set your **mac** and **ip** as necessary, use the **ip** and **mac** that worked from your previous Ethernet shield explorations!

The **card**, **volume**, and **root** are objects that help us traverse the complex structure of an SD card

The error function is not too exciting, it just prints out the error and sits in a **while(1);** loop forever

```
/*
 * This sketch will list all files in the root directory and
 * then do a recursive list of all directories on the SD card.
 *
 */

#include <SdFat.h>
#include <SdFatUtil.h>
#include <Ethernet.h>

/%%**%%%%**%%%%**%%%%**%%%%**%%%% ETHERNET STUFF %%**%%%%**%%%%**%%%%**%%%%**%%%%/
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = { 192, 168, 1, 177 };
Server server(80);

/%%**%%%%**%%%%**%%%%**%%%%**%%%% SDCARD STUFF %%**%%%%**%%%%**%%%%**%%%%**%%%%/
Sd2Card card;
SdVolume volume;
SdFile root;

// store error strings in flash to save RAM
#define error(s) error_P(PSTR(s))

void error_P(const char* str) {
  PgmPrint("error: ");
  SerialPrintln_P(str);
  if (card.errorCode()) {
    PgmPrint("SD error: ");
    Serial.print(card.errorCode(), HEX);
    Serial.print(',');
    Serial.println(card.errorData(), HEX);
  }
  while(1);
}
```

Part 2 is the **setup()** function. It sets up the Serial object so we can debug the connection in real time. It then prints out the RAM usage. You'll need a **Atmega328**Arduino for this experiment, and you should see at leat 1000 bytes of RAM free. Once this gets to under 250 bytes, you may be running too low!

Then we do the trick where we make the hardware **SS** pin #10 to an **OUTPUT** and **HIGH** to disable the wiznet chip while we check the card contents. If you're on a Mega, change this to 53. Then we initialize the card which should go fine since you already tested this before

Then we verify the card structure, print out all the files, and print "Done!". Finally we stuck the Ethernet initialization code at the end here! Now we have both the Ethernet and SD card working

```
void setup() {
  Serial.begin(9600);

  PgmPrint("Free RAM: ");
  Serial.println(FreeRam());

  // initialize the SD card at SPI_HALF_SPEED to avoid bus errors with
  // breadboards.  use SPI_FULL_SPEED for better performance.
  pinMode(10, OUTPUT);                       // set the SS pin as an output (necessary!)
  digitalWrite(10, HIGH);                    // but turn off the W5100 chip!

  if (!card.init(SPI_HALF_SPEED, 4)) error("card.init failed!");

  // initialize a FAT volume
  if (!volume.init(&card)) error("vol.init failed!");

  PgmPrint("Volume is FAT");
  Serial.println(volume.fatType(),DEC);
  Serial.println();

  if (!root.openRoot(&volume)) error("openRoot failed");

  // list file in root with date and size
  PgmPrintln("Files found in root:");
  root.ls(LS_DATE | LS_SIZE);
  Serial.println();

  // Recursive list of all directories
  PgmPrintln("Files found in all dirs:");
  root.ls(LS_R);

  Serial.println();
  PgmPrintln("Done");

  // Debugging complete, we start the server!
  Ethernet.begin(mac, ip);
  server.begin();
}
```

We'll skip ahead to the **loop()** where we wait for clients (checking via **server.available()**) and then read the client request before responding. This is basically copy-and-pasted from the **Webserver** example sketch that comes with the Ethernet library (well, the first and last parts of the loop are at least).

There's a little trick where to simplify the code, the writer of this sketch doesn't actually check to see what file the web browser wants, it always spits out the same thing. In this case, we're going to have it spit out the files by using a helper function called **ListFiles(client, 0)** which we skipped over but will show next. The 0 in the second argument to the function just tells the function whether to print out the file sizes

```
void loop()
{
  Client client = server.available();
  if (client) {
    // an http request ends with a blank line
    boolean current_line_is_blank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        // if we've gotten to the end of the line (received a newline
        // character) and the line is blank, the http request has ended,
        // so we can send a reply
        if (c == '\n' && current_line_is_blank) {
          // send a standard http response header
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println();

          // print all the files, use a helper to keep it clean
          //ListFiles(client, 0);
          client.println("<h2>Files:</h2>");
          ListFiles(client, 0);

          break;
```

```
        }
        if (c == '\n') {
          // we're starting a new line
          current_line_is_blank = true;
        } else if (c != '\r') {
          // we've gotten a character on the current line
          current_line_is_blank = false;
        }
      }
    }
    // give the web browser time to receive the data
    delay(1);
    client.stop();
  }
}
```

Now we'll go back and exampine the **ListFiles** function. This is a bit tedious, but worth looking at. We've simplified it by removing recursive listing, which means we don't list files in any subdirectories.

The **dir_t p** object is a "Directory Entry" holder. It will store the information for each entry in the directory.

We first reset the root directory by **rewind()**'ing it. Then we read the directory file by file. Some files are unused or are the "." and ".." (up directory) links, which we ignore. We also only list FILEs or SUBDIRectories.

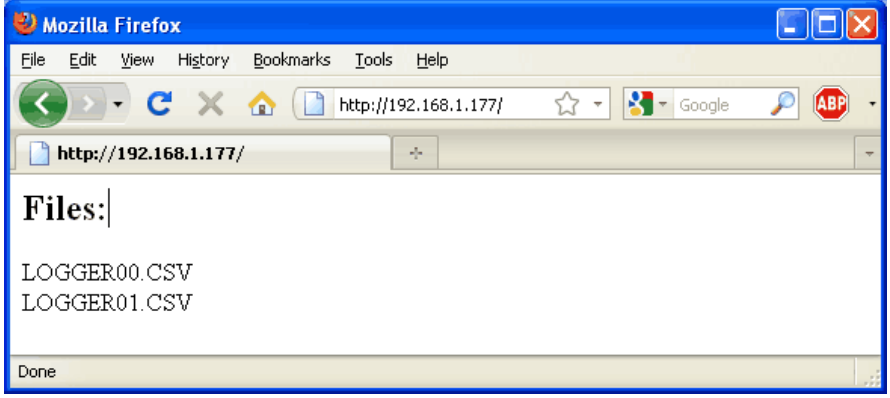
Then we print the name out by going through all 11 characters (remember the file names are in 8.3 format) and ignore the space. We also stick the '.' between the first 8 and last 3 characters.

If its a directory type file, we put a slash at the end to indicate it. If its not, we can print out the file size in bytes.

Finally, after each file name we stick in a "
" which will go to the next line in a web browser

```
void ListFiles(Client client, uint8_t flags) {
  // This code is just copied from SdFile.cpp in the SDFat library
  // and tweaked to print to the client output in html!
  dir_t p;

  root.rewind ();
  while (root.readDir(p) > 0) {
    // done if past last used entry
    if (p.name[0] == DIR_NAME_FREE) break;

    // skip deleted entry and entries for . and  ..
    if (p.name[0] == DIR_NAME_DELETED || p.name[0] == '.') continue;

    // only list subdirectories and files
    if (!DIR_IS_FILE_OR_SUBDIR(&p)) continue;


    // print file name with possible blank fill
    //root.printDirName(*p, flags & (LS_DATE | LS_SIZE) ? 14 : 0);


    for (uint8_t i = 0; i < 11; i++) {
      if (p.name[i] == ' ') continue;
      if (i == 8) {
        client.print('.');
      }
      client.print(p.name[i]);
    }
    if (DIR_IS_SUBDIR(&p)) {
      client.print('/');
    }

    // print modify date/time if requested
    if (flags & LS_DATE) {
       root.printFatDate(p.lastWriteDate);
       client.print(' ');
       root.printFatTime(p.lastWriteTime);
    }
    // print size if requested
    if (!DIR_IS_SUBDIR(&p) && (flags & LS_SIZE)) {
      client.print(' ');
      client.print(p.fileSize);
    }
    client.println("<br>");
  }
}
```

OK after all that work, lets upload that sketch to the Arduino! Make sure you have the correct **ip** address for your network, then use a browser on the same network to visit your website. Here is what I got - there are those two files from my previous datalogging experiments!



Browsing files!

Obviously, we should make it so you can clicky those file names, eh? Well! Thats the next sketch, you can download the latest version from github here [http://github.com/adafruit/SDWebBrowse] (click Download Source) in the top right hand corner!

Fix the **ip** address to match your network, and SS pin (if you're on a Mega)

Not a lot has changed between the previous code, the setup is the same, the big changes are in the **loop()** code. The bones are the same - we look for new client connections. But this time we read the client request into a character buffer (**clientline**) until we get a newline character (such as \n or \r). This indicates we have read an entire line of text. To 'finish' the string, we put a null character (0) at the end.

We then use **strstr** which will look for substrings. If we have a "GET / HTTP" request for the root directory, we do the same as before, printing out the list of files.

If we have no space after "GET /" that means its something like "GET /file" which means we will have to extract the filename. We make a pointer to the string and start it right after the first slash. Then we look for the beginning of the "HTTP/1.1" string which follows the filename request and turn the first character into a string-terminator. Now we have the name of the file which we can try to open.

If we fail to open the file, we will return a **404**. Otherwise, we print out all of the file contents.

```
// How big our line buffer should be. 100 is plenty!
#define BUFSIZ 100

void loop()
{
  char clientline[BUFSIZ];
  int index = 0;

  Client client = server.available();
  if (client) {
    // an http request ends with a blank line
    boolean current_line_is_blank = true;

    // reset the input buffer
    index = 0;

    while (client.connected()) {
      if (client.available()) {
        char c = client.read();

        // If it isn't a new line, add the character to the buffer
        if (c != '\n' && c != '\r') {
          clientline[index] = c;
          index++;
          // are we too big for the buffer? start tossing out data
          if (index >= BUFSIZ)
            index = BUFSIZ -1;

          // continue to read more data!
          continue;
        }

        // got a \n or \r new line, which means the string is done
        clientline[index] = 0;

        // Print it out for debugging
        Serial.println(clientline);

        // Look for substring such as a request to get the root file
        if (strstr (clientline, "GET / ") != 0) {
          // send a standard http response header
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println();
```

```
        // print all the files, use a helper to keep it clean
        client.println("<h2>Files:</h2>");
        ListFiles(client, LS_SIZE);
      } else if (strstr (clientline, "GET /") != 0) {
        // this time no space after the /, so a sub-file!
        char *filename;

        filename = clientline + 5; // look after the "GET /" (5 chars)
        // a little trick, look for the " HTTP/1.1" string and
        // turn the first character of the substring into a 0 to clear it out.
        (strstr (clientline, " HTTP"))[0] = 0;

        // print the file we want
        Serial.println(filename);

        if (! file.open(&root, filename, O_READ)) {
          client.println("HTTP/1.1 404 Not Found");
          client.println("Content-Type: text/html");
          client.println();
          client.println("<h2>File Not Found!</h2>");
          break;
        }

        Serial.println("Opened!");

        client.println("HTTP/1.1 200 OK");
        client.println("Content-Type: text/plain");
        client.println();

        int16_t c;
        while ((c = file.read()) > 0) {
            // uncomment the serial to debug (slow!)
            //Serial.print((char)c);
            client.print((char)c);
        }
        file.close();
      } else {
        // everything else is a 404
        client.println("HTTP/1.1 404 Not Found");
        client.println("Content-Type: text/html");
        client.println();
        client.println("<h2>File Not Found!</h2>");
      }
      break;
    }
  }
  // give the web browser time to receive the data
  delay(1);
  client.stop();
 }
}
```

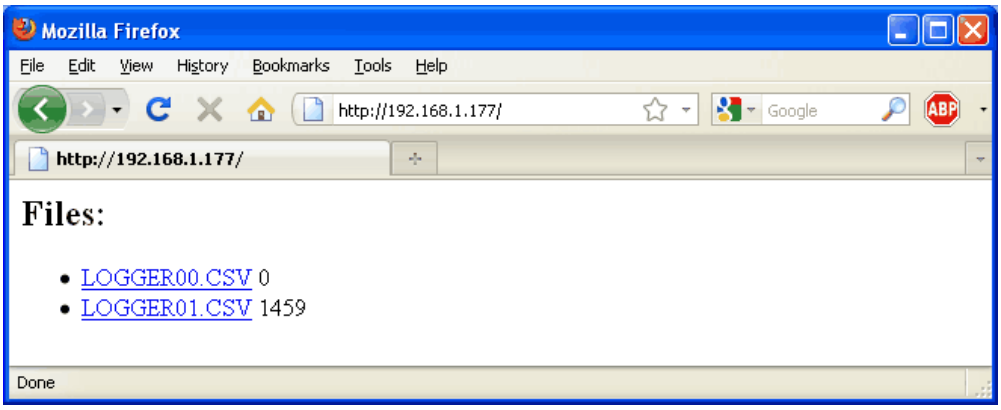
Lets look at the new file listing code as well, its very similar, but now we've added a bit of HTML so that each file is part of a **<ul>** list and the name is a URL link.

```
void ListFiles(Client client, uint8_t flags) {
  // This code is just copied from SdFile.cpp in the SDFat library
  // and tweaked to print to the client output in html!
  dir_t p;

  root.rewind ();
  client.println("<ul>");
  while (root.readDir(p) > 0) {
    // done if past last used entry
    if (p.name[0] == DIR_NAME_FREE) break;

    // skip deleted entry and entries for . and  ..
    if (p.name[0] == DIR_NAME_DELETED || p.name[0] == '.') continue;

    // only list subdirectories and files
    if (!DIR_IS_FILE_OR_SUBDIR(&p)) continue;

    // print any indent spaces
    client.print("<li><a href=\"");
    for (uint8_t i = 0; i < 11; i++) {
      if (p.name[i] == ' ') continue;
      if (i == 8) {
        client.print('.');
      }
      client.print(p.name[i]);
    }
    client.print("\">");
```

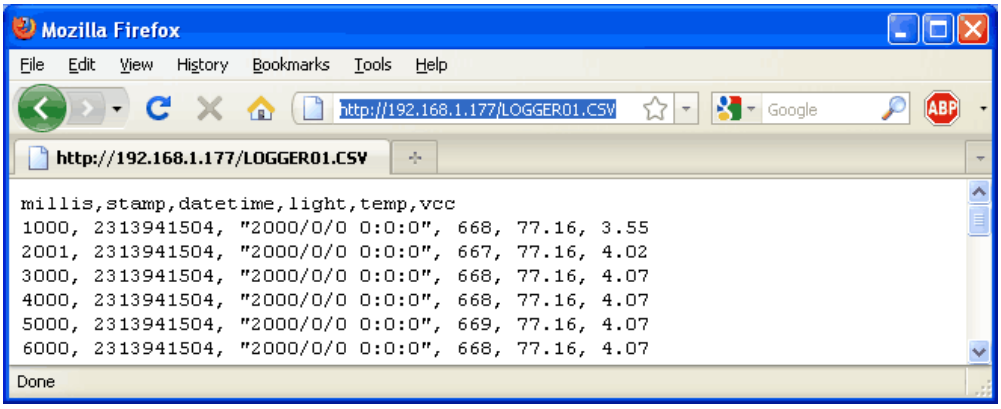
```
    // print file name with possible blank fill
    for (uint8_t i = 0; i < 11; i++) {
      if (p.name[i] == ' ') continue;
      if (i == 8) {
        client.print('.');
      }
      client.print(p.name[i]);
    }

    client.print("</a>");

    if (DIR_IS_SUBDIR(&p)) {
      client.print('/');
    }

    // print modify date/time if requested
    if (flags & LS_DATE) {
       root.printFatDate(p.lastWriteDate);
       client.print(' ');
       root.printFatTime(p.lastWriteTime);
    }
    // print size if requested
    if (!DIR_IS_SUBDIR(&p) && (flags & LS_SIZE)) {
      client.print(' ');
      client.print(p.fileSize);
    }
    client.println("</li>");
  }
  client.println("</ul>");
}
```

OK upload the sketch already!

Now you'll see that the file names have turned into links (we also added the file size in bytes)



Click on a filename to view it



If you try to view a file that is not on the card, you'll get a 404 error (file not found)