Application Note - Web Server with W5100 device and SD-Card

Author:
Joel - MyFreescaleWebPage
http://myfreescalewebpage.free.fr

Last revision of this document: 1.1 of 2012-01-19.

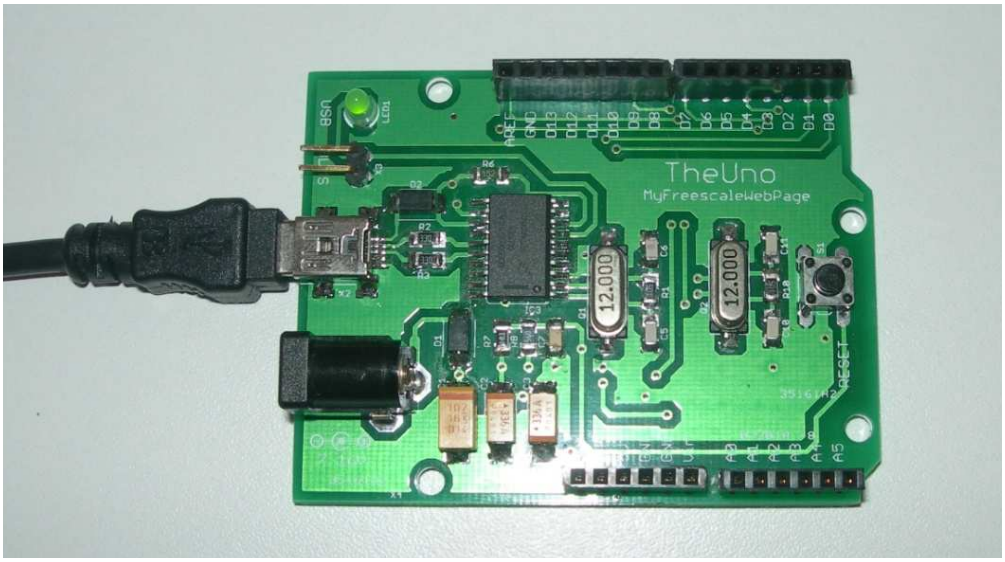
Table of contents

Introduction

A web server can be very useful in many applications to monitor statuses or to easily modify some parameters: whether stations, home automation systems, etc. Moreover, it is possible today to access it throw the Internet, using any computer or Smartphone.

This application note shows how to simply implement a web server on a very small microcontroller. It uses the W5100 Ethernet controller from Wiznet to establish the network connection and a simple SD-Card for files storage. Two DS18B20 temperature sensors will be used for demonstration.

This application note has been developed on TheUno, a development board created by MyFreescaleWebPage, based on S08DZ60 device and with a build-in open-source debugging cable.



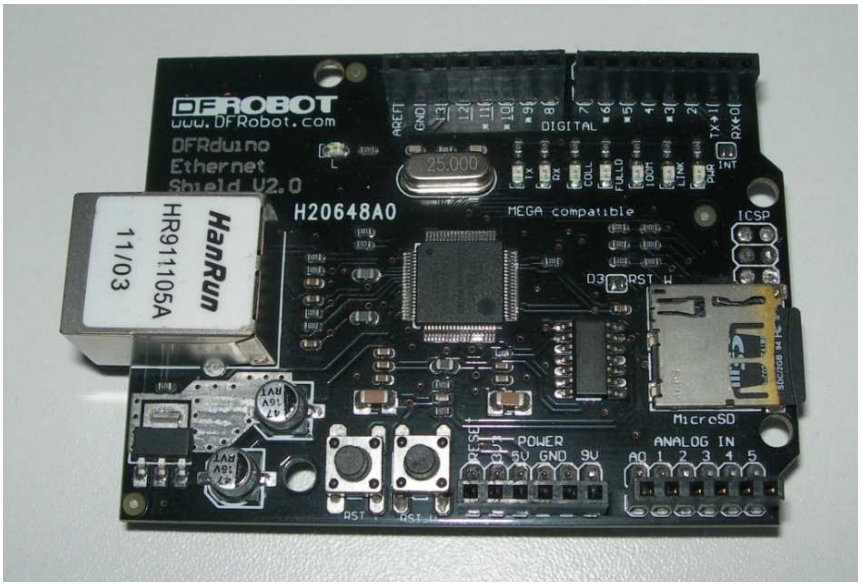
TheUno is compatible with almost every shields designed for the Arduino. It is possible to plug multiple shields simultaneously on the development board, with Ethernet, Wi-Fi, GPS, audio devices, H-Bridge, ... to simply create powerful applications using Freescale CodeWarrior, a nice development environment based on Eclipse. More details about TheUno available on http://myfreescalewebpage.free.fr.

1 Hardware

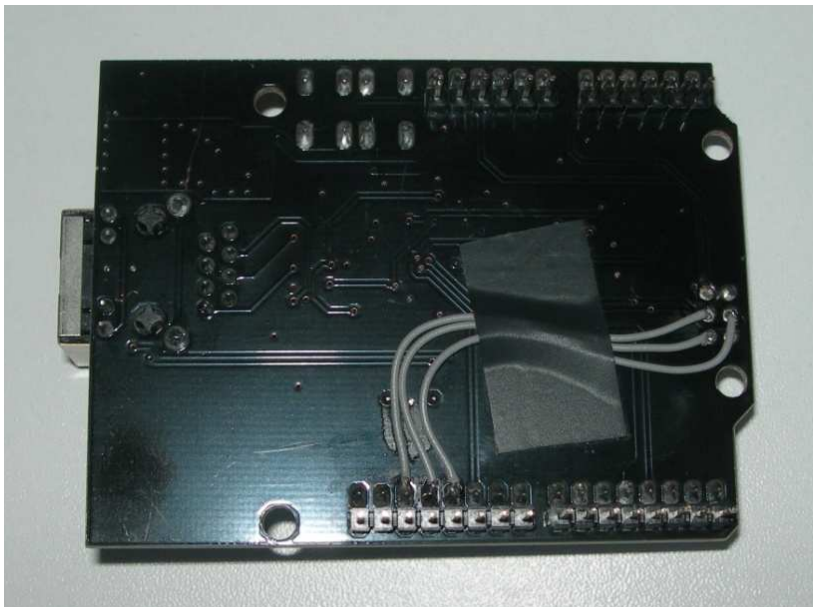
1.1 The Ethernet shield

The shield chosen for this application note comes from DFRobot, and is available on the following page: http://www.dfrobot.com/index.php?route=product/product&path=35_39&product_id=455.

This shield is based on the W5100 Ethernet controller from Wiznet, which is very easy to use thanks to its SPI interface. The shield also contains a Micro-SD card connector, making this board so interesting to build a web server.



However, the SPI interface on the board is connected using the ICSP connector of the Arduino, and not the SPI pins D11-D13. In order to be used with TheUno, the ICSP connector must be removed and replaced with some wiring, as shown on the following photo.



After this simple modification, we get:
- MOSI on D11 (MOSI pin of the S08DZ60 SPI peripheral);
- MISO on D12 (MISO pin of the S08DZ60 SPI peripheral);
- SCK on D13 (SCK pin of the S08DZ60 SPI peripheral);
- W5100 selection (active low) on D10 (PTE2 on the S08DZ60 microcontroller);
- SD-Card selection (active low) on D4 (PTA0 on the S08DZ60 microcontroller).

1.2 The temperature sensors

Two DS18B20 temperature sensors will be used for demonstration purpose. They are connected to A0 (PTA6) and A1 (PTA5) according to the following schematic.

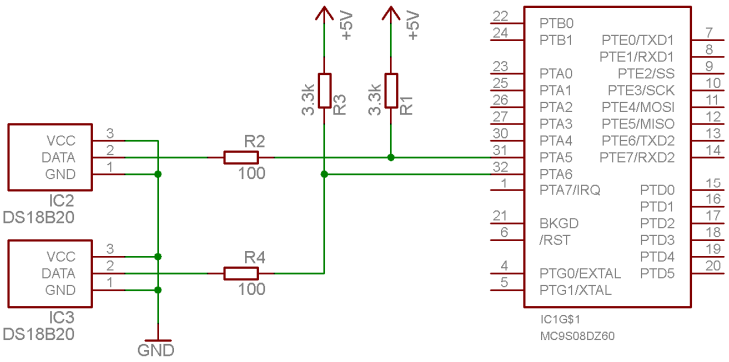
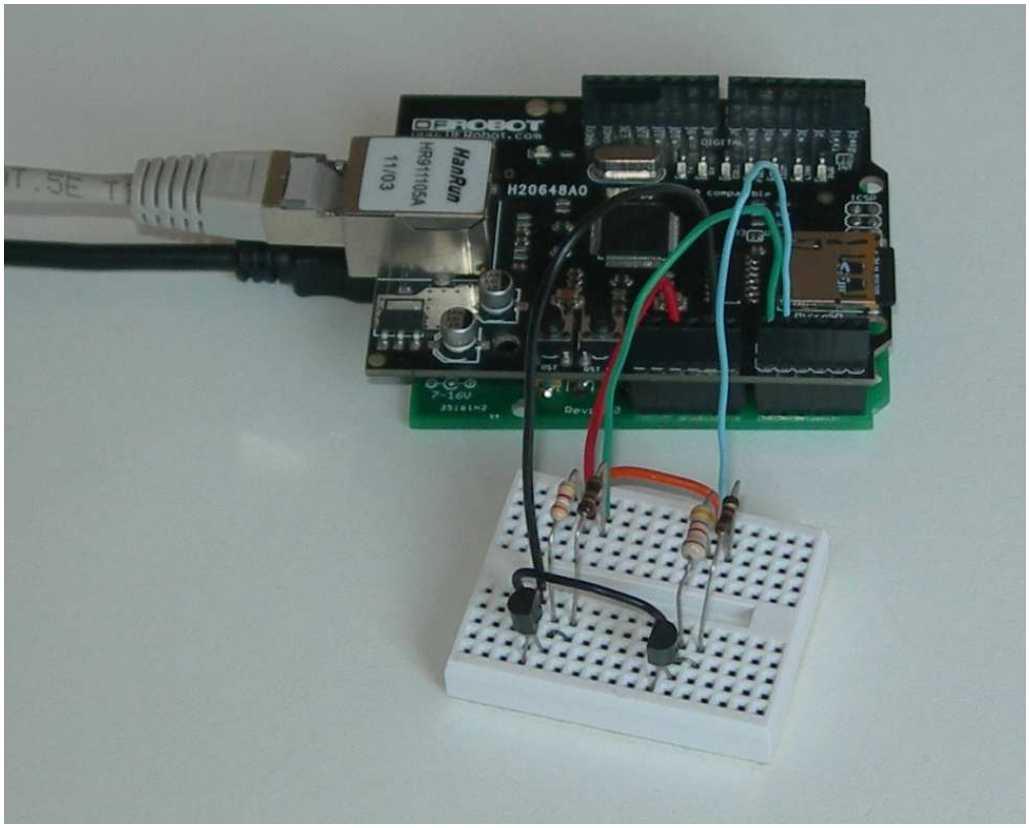


Figure 1: Connection of DS18B20 sensors

The sensors are connected using a simple breadboard.



We are now ready to develop the web server application.

2 Software architecture

The software architecture of the web server is shown on the following diagram.

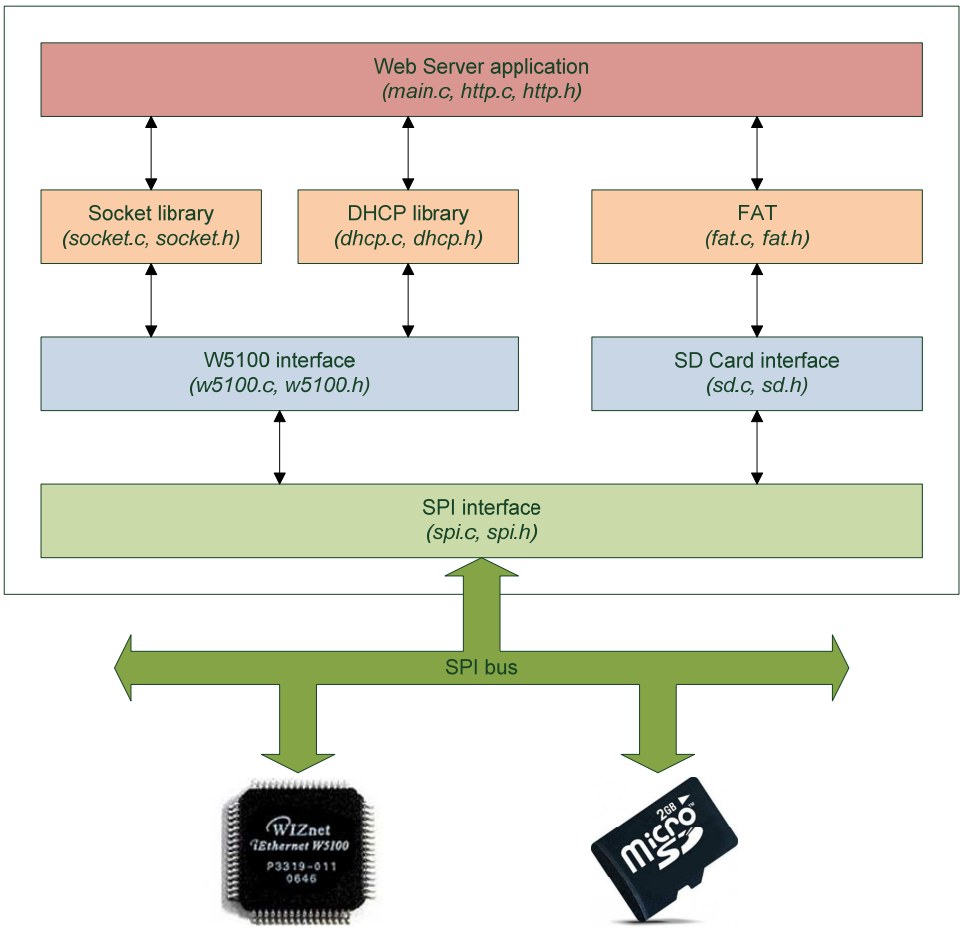


Figure 2: Software architecture

We are now going to examine the different parts of the software to understand how the web server application is working.

2.1 Description of the libraries

2.1.1 SPI interface

The SPI interface contains all the definitions and functions used to communicate with the components on the SPI bus (the W5100 and the SD-Card). It can be modified to match the hardware requirements, for example to modify the pinout or to add devices on the SPI bus. The goal of the SPI module is to get low-level W5100 and SD-Card libraries which are not hardware dependant.

Some general functions are available to communicate with the SPI components:
- SPI_Init_HighRate() to initialize the interface;
- SPI_Write(), SPI_Read() and SPI_WriteAndRead() to exchange data.

The functions SPI_SelectW5100Device(), SPI_UnselectW5100Device(), SPI_SelectSDCard() and SPI_UnselectSDCard() are used to select and unselect the W5100 Ethernet controller and the SD-Card.

Some specific functions to the SD-Card are written to get its status: SPI_IsSDCardPresent() to know if the SD-Card is inserted and SPI_IsSDCardWriteProtected() to get the lock switch position. Because those signals are not available on the DFRobot shield which has been chosen in this application note, those functions simply returns default values. They can be modified if you are using another board on which those signals are available.

Finally, because the SD-Card initialization requires 375kHz SPI clock, a specific function called SPI_Init_LowRate() has been written to get this specific clock frequency on the SPI interface. This function is written according to the MCU clock (18MHz in this example, explaining a prescaler factor set to 48 in the source code).

2.1.2 W5100 low-level interface

The W5100 low-level interface contains the definitions of the W5100 registers and the W5100 configuration parameters which can be modified depending of the application which is developed: sockets size, MAC address, IP Address, Mask, Gateway IP Address, use of DHCP.

The W5100 interface also contains the specific functions used to communicate with the W5100 device throw the SPI bus:
- W5100_Init() to initialize the device;
- W5100_WriteByte() and W5100_ReadByte() to exchange data throw the SPI interface.

W5100_WriteByte() and W5100_ReadByte() functions are written according to the W5100 datasheet, by sending the write or read command (0xF0 and 0x0F) and followed by the address where to write or read the data byte in the W5100 memory map. This is shown on the following schemas.

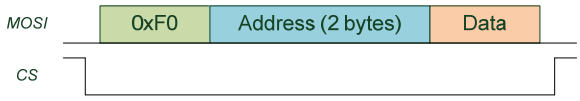


Figure 3: Writing to W5100 device

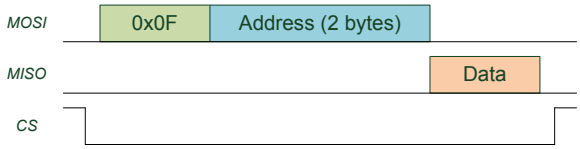


Figure 4: Reading from W5100 device

2.1.3 W5100 libraries

Two W5100 libraries are provided today but additional libraries can be written depending of the needs of your own applications (for example if you need DNS resolution, etc).

2.1.3.1 Socket library

The socket library is used to manipulate sockets. Several protocol (TCP, UDP...) are available. Moreover, both server and client applications can be created with the functions available in this library.

The following functions are available:
- SOCKET_Open() to open a new TCP, UDP, IPRAW, MACRAW or PPPOE socket;
- SOCKET_Close() to close a socket;
- SOCKET_Connect() to establish a connection with a remote TCP server;
- SOCKET_Disconnect() to terminate a connection;
- SOCKET_GetStatus() to get socket status from W5100 device;
- SOCKET_Listen() to listen for client connection;
- SOCKET_Send() and SOCKET_SendTo() to send data;
- SOCKET_Recv() and SOCKET_RecvFrom() to get data.

For example, a simple TCP Client application can be created as shown below.

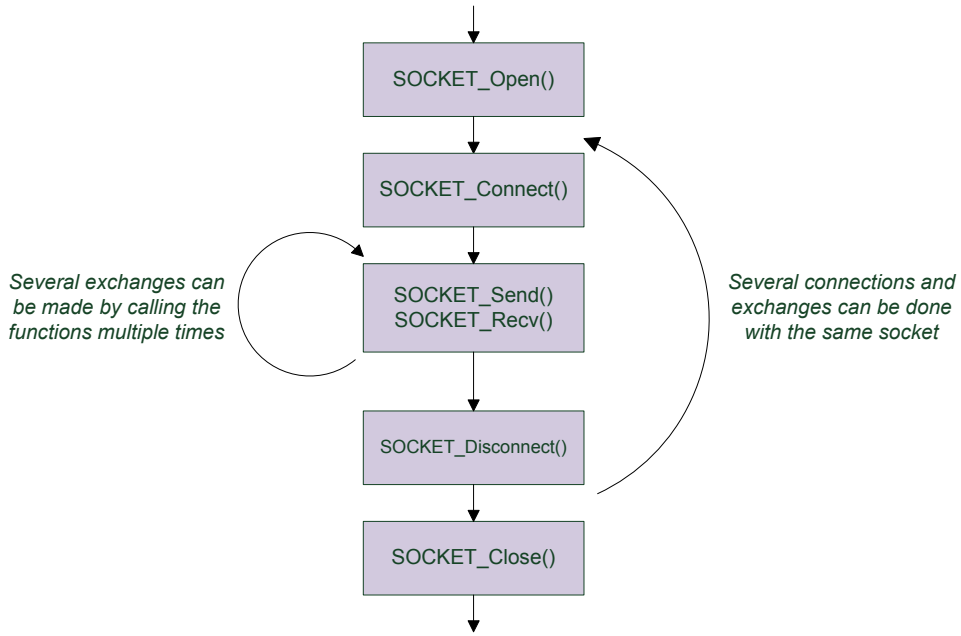


Figure 5: TCP Client application implemented using the W5100 socket library

2.1.3.2 DHCP library

The DHCP library provides a single function DHCP_GetIPAddress() to initialize the W5100 device using DHCP. It uses the socket library to send and receive messages on the network.

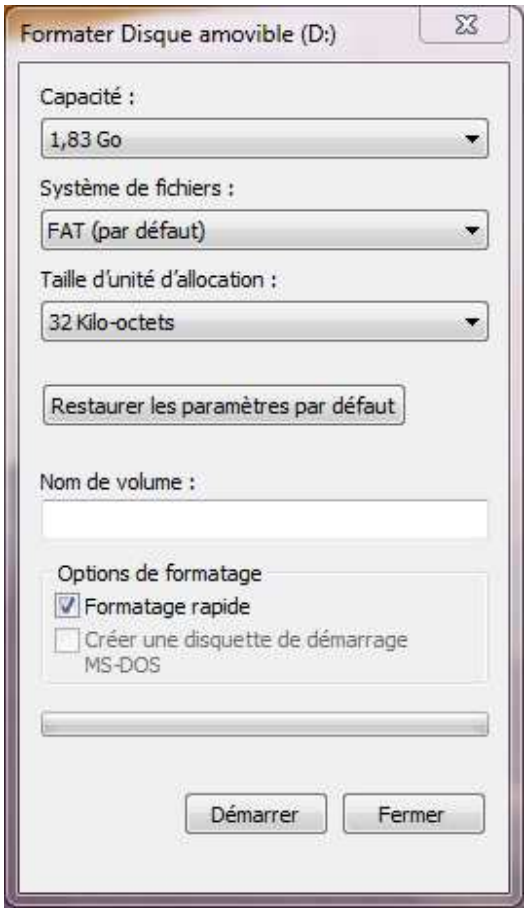
2.1.4 SD-Card low-level interface

The SD-Card low-level interface contains the definitions and functions used to communicate with the SD-Card throw the SPI bus.

A successful call to SD_Init() must be done before using the SD-Card. Remember that the SD-Card initialization must be performed using a 375kHz SPI clock (use of SPI_Init_LowRate() function). After that, the functions SD_ReadBlock() and SD_WriteBlock() are called to read and write in the SD-Card.

Please note that the SD-Card low-level interface does not support hot insertion and removing of the SD-Card. The card must be inserted before powering up the microcontroller and kept in place for proper operation.

Finally, please format the SD-Card before any use on the microcontroller. Default formatting settings can be used as shown on the following screenshot (2GB SD-Card formatted on Windows 7 computer).



2.1.5 FAT library

The file system of the SD-Card is managed by the FAT library.

Before any use of the SD-Card, the initialization of the FAT library is done calling the function FAT_ReadBootSector(). This function reads the SD-Card Boot Sector, in which several useful information are registered (Cluster size, Sector size, etc).

The FAT library contains several basic functions to read and write files on the SD-Card: FAT_FileOpen(), FAT_FileRead(), FAT_FileWrite(), FAT_FileClose().

Reading a file is very simple and is done as shown on the following schema.

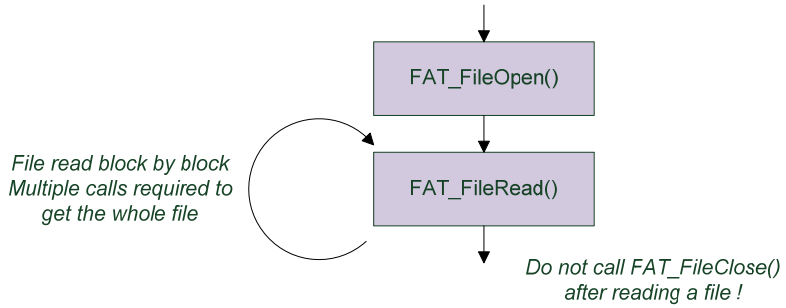


Figure 6: Reading a file using the FAT library

FAT_FileOpen() returns an error if the file is not found on the SD-Card. The file is read block by block. In order to get all the file content, FAT_FileRead() returns the number of bytes read and can be called until it returns zero.

Creating or modifying a file is also very easy.

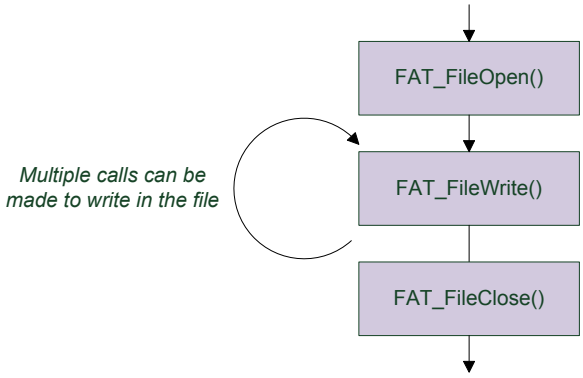


Figure 7: Writing or modifying a file using the FAT library

FAT_FileOpen() returns an error if it is not possible to create or modify the file. The function FAT_FileWrite() add data at the end of the file. It is possible to call it several times before closing the file. Finally, FAT_FileClose() must be called to close the file.

Please note that the library does not support directories and is limited to 8.3 filename format.

2.2 The Web Server application

The web server application uses the W5100 and FAT libraries. It simply opens a new TCP socket on port number 80 and waits for a client connection. The web server replies to the requests made by clients.

2.2.1 Requests and replies

The goal here is not to explain the whole HTTP protocol, but simply to show a single exchange between a browser and the web server. If you want to have a look on some exchanges on your own computer, you can use Wireshark software, available on the following website: http://www.wireshark.org.

We simply imagine that your browser is calling for a page named "example.htm". A connection is established with the server and the following request is sent to get the content of the page:

```
GET /example.htm HTTP/1.1\r\n
.....
```

The first line contains the "GET" command, the filename of the page requested and the HTTP protocol version. The following lines, not shown above, contain several additional information about the client.

The server receives this request and formats a new response with the content of the file "example.htm":

```
HTTP/1.0 200 OK\r\n
Pragma: no-cache\r\n
Expires: Fri, 01 Jan 1990 00:00:00 GMT\r\n
Cache-Control: no-cache, must-revalidate\r\n
Content-Type: text/html\r\n
Content-Length: 100\r\n
\r\n
Content_of_the_file_example_htm\r\n
```

The server is not using cache control, and so simply indicate that the file must be reloaded each time the browser want to display it. The server indicate the length of the file (100 bytes in this example) and send the content of the file. The reply can be sent in multiple packets.

The same mechanism is used to send images, JavaScript files, etc. Content type in the reply is set according to the file extension.

The web server application attached to this application note is able to manage the following files: *.htm, *.js, *.bmp, *.ico, *.jpeg, *.gif, *.png. Additional extensions can be easily added to the application.

2.2.2 Web Server architecture

The web server is implemented using two simple functions:
- "HTTP_Init()", called on start-up to perform the web server initializations (W5100 device, SD-Card and FAT library).
- "HTTP_Server()", called periodically to perform the main job of the web server: getting client requests and sending responses.

The functions are shown on the following schema.

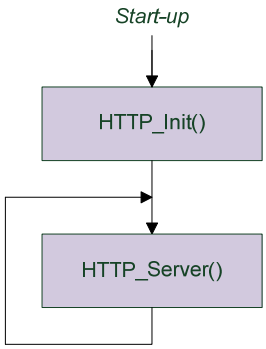


Figure 8: Calls of the web server functions

The function "HTTP_Server()" is built using a state machine based on the web server socket status, as shown below.

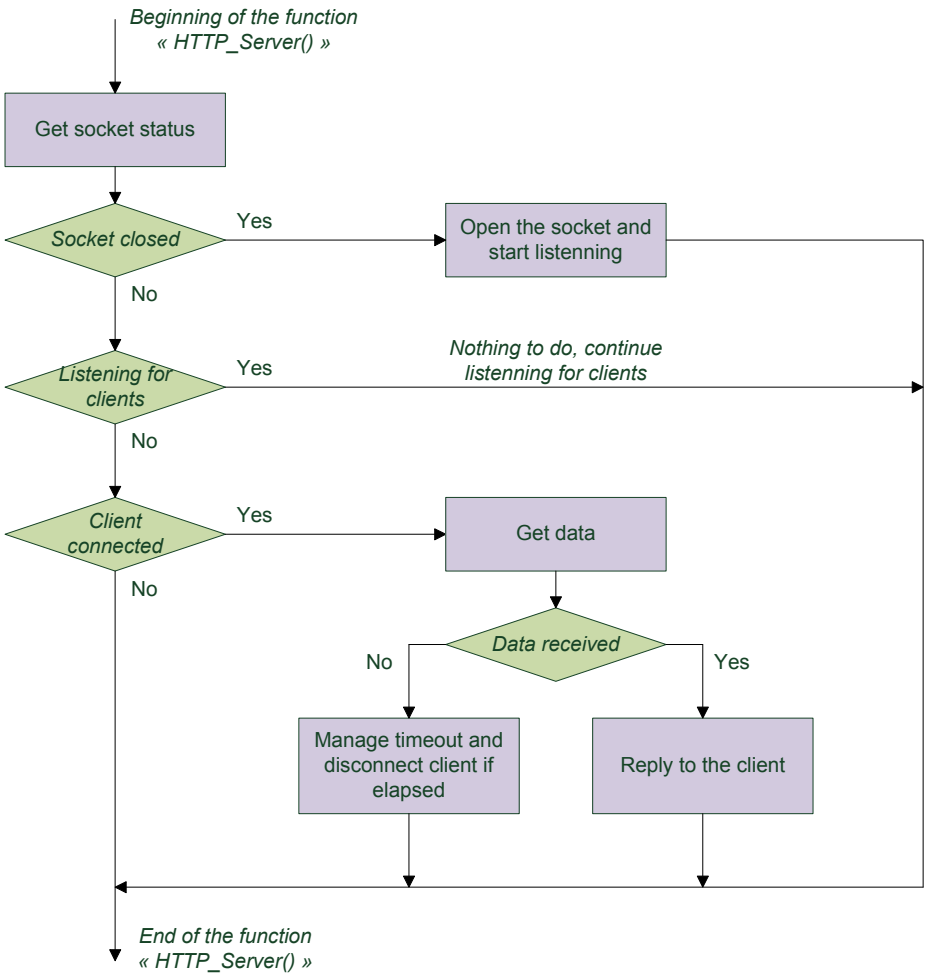


Figure 9: Client requests management

The socket status is get from the W5100 device. Three statuses are particularly interesting:
- If the socket is closed, it is opened and initialized to listen for a new client;
- When the socket is listening for clients, the web server has nothing to do;
- Finally, if a client is connected, the web server gets the data received and replies. The connection is kept alive for a few moment, permitting the client to send a new request without the need to open a new connection. The socket is closed after a small timeout.

You can notice that only a single client can be connected at the same time on the web server. It is also why the socket is closed quickly after some exchanges, in order to enable several browsers to get data from the web server.

3 DS18B20 Thermometer example

The goal of this example is to display the temperature of two DS18B20 sensors on a simple web page.

In order to get a nice result, a JavaScript library will be used to create a dynamic chart. This library is HighCharts, available on http://www.highcharts.com. Several examples are shown on this website for demonstration purposes. In this application note, I will draw a simple chart with the temperature of the DS18B20 sensors periodically refreshed.

3.1 Content of the SD-Card

To get this application working, the SD-Card contains several files:
- "index.htm" is the main page on which the DS18B20 temperatures will be displayed;
- "great.jpg" is an image displayed on the index page;
- "jq142.js" is JQuery library (V1.4.2);
- "hc125.js" is HighCharts library (V1.2.5);
- "hcblue.js", "hcgray.js", "hcgreen.js", "hcgrid.js" and "hcskies.js" are used to personalized the HichCharts style;
- "favicon.ico": is the favicon (image displayed on the left of the website address in your browser).

Please note that the filenames are chosen according to the limitations of the FAT library (no directories and 8.3 filename format).

3.2 Creation of the chart

When the page "index.htm" is loaded from the web server, it creates a simple chart using the HighCharts library:

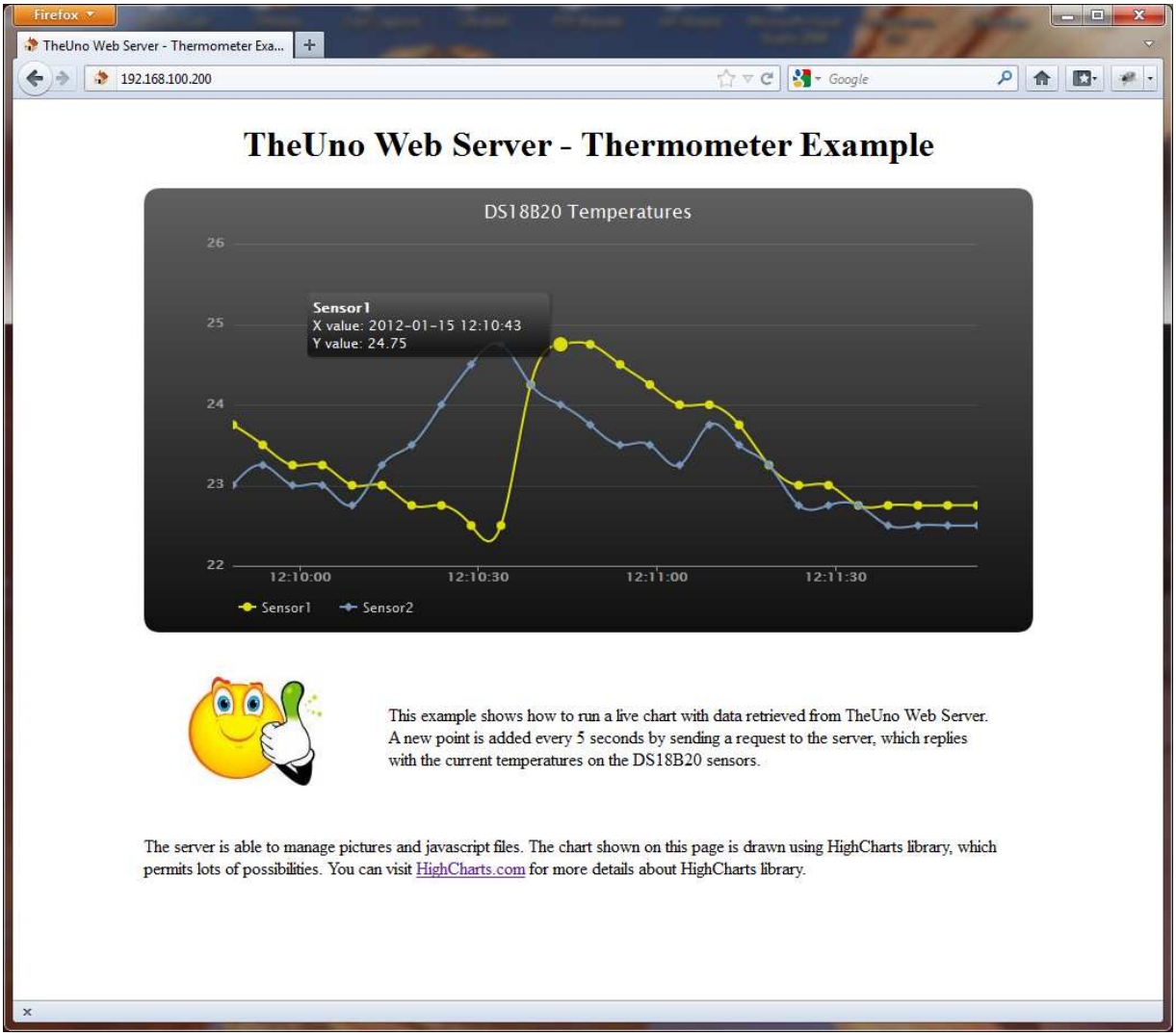
```
$(document).ready(function()
{
      Chart = new Highcharts.Chart(......);
});
```

Several initializations are made and can be modified. More particularly, the function setInterval() is called to get a periodical refresh of the chart:

```
setInterval(GetTemperature, 5000); /* GetTemperature called every 5s */
```

The function GetTemperature() is written in the file "index.htm". This function simply sends an HTTP request "GetTemperature" to the web server, gets the answer which is containing the two DS18B20 temperatures, and adds the data on the chart.

It gives the following result:



For more details, please, refer to the code of the web page "index.htm".

3.3 Web Server specific coding

As previously explained, an HTTP request "GetTemperature" is sent to get the DS18B20 temperatures from the web server. This request looks like:

```
GET /GetTemperature HTTP/1.1\r\n
.....
```

This is quite the same request than the one sent to get a file on the SD-Card, but it is called "GetTemperature", without any extension. A specific coding in the function HTTP_Server() recognizes this request and performs the following reply:

```
HTTP/1.0 200 OK\r\n
Pragma: no-cache\r\n
Expires: Fri, 01 Jan 1990 00:00:00 GMT\r\n
Cache-Control: no-cache, must-revalidate\r\n
Content-Type: text/html\r\n
\r\n
22.0;21.75\r\n
```

In this reply, the temperatures of the sensors are 22.0°C and 21.75°C.

Conclusion

This document gives the main explanations to use the attached web server application in your own projects, with any 8 to 32bits microcontroller and the W5100 Ethernet controller.

Of course some improvements can be done on this example. The main one is the management of the cache to not reload the files from the server each time a page is opened on your web browser. To get it, the FAT library must be modified to retrieve the date of the last modifications on the file, and the web server functions must be improved in order to correctly reply to the browser when a file is requested.

Finally, note that if you want to access the web server over the Internet, you simply have to forward the port number 80 in your router. In this case, do not forget to make a DHCP reservation to always get the web server to the same IP address on your local network.