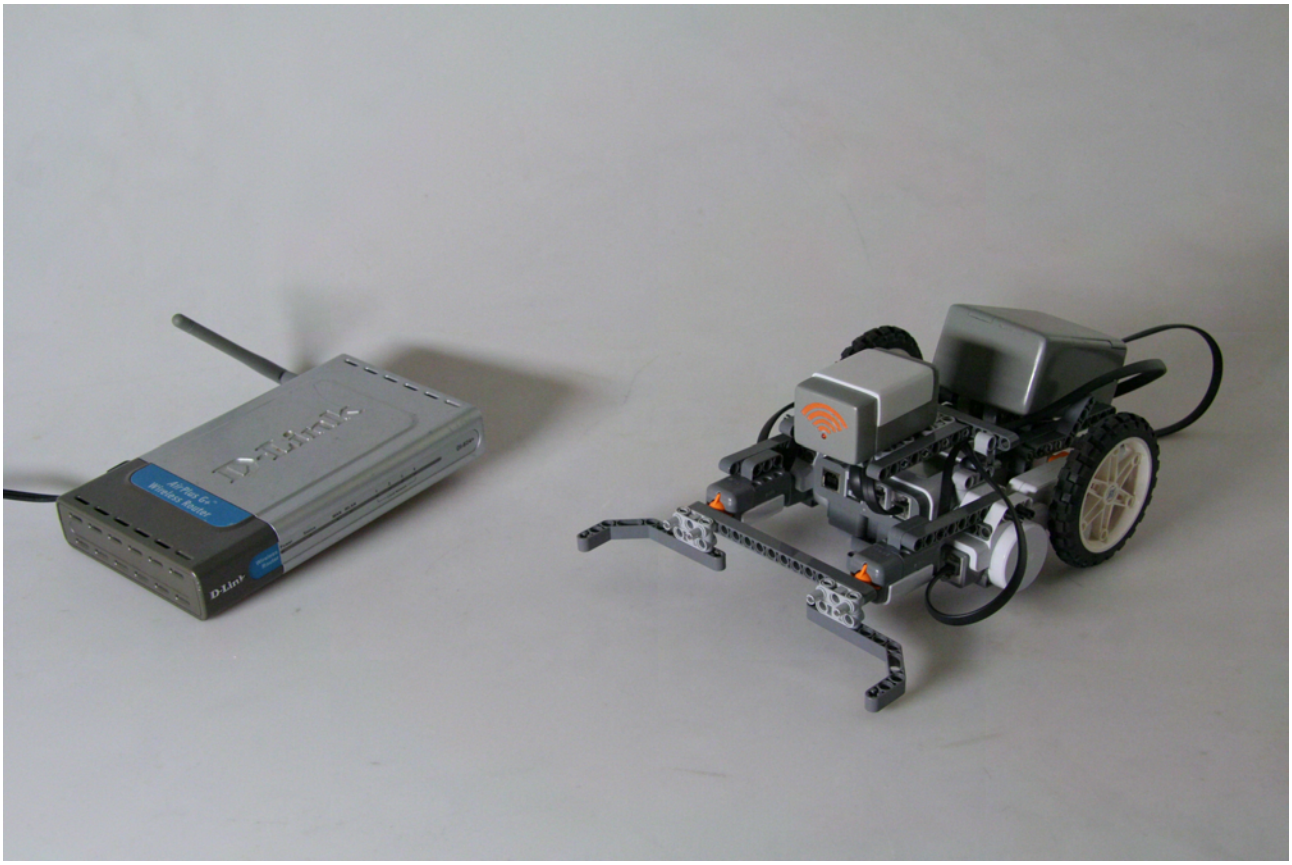
Techno Class process report:

Lego beyond toys

connect lego NXT to the internet



Sjef Fransen

j.g.m.fransen@student.tue.nl

4 December 2009

Table of contents

Introduction

This report described in detail the lego extension and explains the concept, the mechanical design, the electrical design, the software and the application. With this report you should be able to get an understanding of the extension pack and use it for your own applications with the Lego NXT.

The WiFiBlock extension

Within the Lego Beyond toys class I have designed the WifiBlock extension kit. This kit consist out of hardware, software and application examples. This report will be a guide to get a better understanding of the extension and how it can be implemented into your designs. The first section will briefly design the concept of the WifiBlock extension. The second section will cover the hardware part and how to build one yourself. The third section will explain the different software parts and the lejos class I have written. The last part will consist out of examples to get started quickly.

Concept

Currently the internet is everywhere. Everybody has a connection to the internet and in the modern world it is almost impossible to live without it. Storing information in databases and getting information out of it at a different location is one of the principles of the internet. This is not only valuable for humans, but robots can benefit form it as well.

By connecting the NXT to the internet new possibilities start to emerge. Instead of giving pre programmed information to the robot, it can look for information itself on the internet. For example it can find a route of which to follow. Next to that it is also able to store data directly into a database without having it to load the file on the computer manually.

My lego extension will be a unit that allows the NXT to get data from and send data to the internet. Besides sending and receiving information, the extension also enables users to control the robot remotely.The extension is simple to use and is general applicable. I have written a special java class “wifiBlock” that can be used within Lejos. The hardware of the extension has an wireless internet connection on board to enable the NXT to move freely.

The Hardware

NXT to Arduino to wiShield

To create a wireless connection to the internet there are multiple technologies available. GPRS, UMTS, 3G and WiFi. The Wifi protocol needs a local access point, while the others connect directly to a satellite. The benefit of WiFi is that it is much faster than the other technologies and while Lego is used most often as indoor equipment a WiFi connection could be easily made. Therefor I have chosen to use WiFi as the communication protocol for the internet.

Connecting a wifi module to the NXT is quite difficult. The NXT has two communication protocols onboard: I2C and RS485. One of these protocols should be used to communicate to a wifi module. Most wifi modules currently available on the market use an SPI interface. To translate the SPI interface towards an I2C protocol I have chosen to use the Arduino environment. The following scheme displays the setup of the wifi connection.

Figure 1: hardware setup of wifiBlock Extension

NXT and Arduino

Although previous students in the class had experimented with Arduino and NXT, they only had one way communication. A sensor was connected to the Arduino and that send the sensor data to the NXT. Within the application the NXT needs to send data to the wifiBlock as well. For the hardware there is no difference in having one way or two way communication. I experimented with both the RS485 and the I2C protocol but eventually the I2C protocol was easier to implement. Furthermore it requires less hardware. The following scheme displays the connections pins of the NXT to the Arduino.

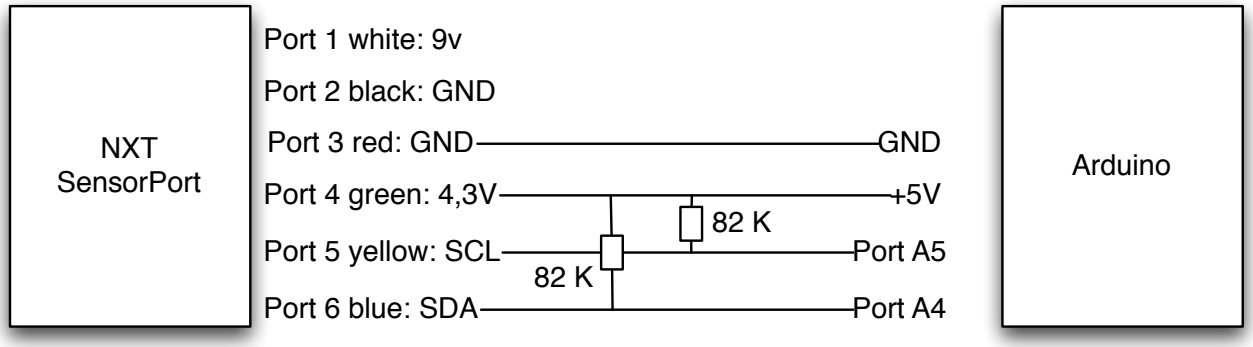


Figure 2: connector scheme NXT-Arduino

Module and Arduino

After fixing the two way communication between the Arduino and the NXT it was time to fix the communication between the Arduino and the Wifimodule. I started to experiment with the wiShield from AsyncLabs that can be directly placed on top of the Arduino Duemilonove.

The wiShield comes with an existing library for Arduino which enables the wiShield to be both server as well as client. This wiShield library can be found at the wiki of Asynclabs and is an open source solution. The combination Arduino - wiShield turned out to be very effective. Figure 3 displays an Arduino Duemilonove with an wiShield on top.

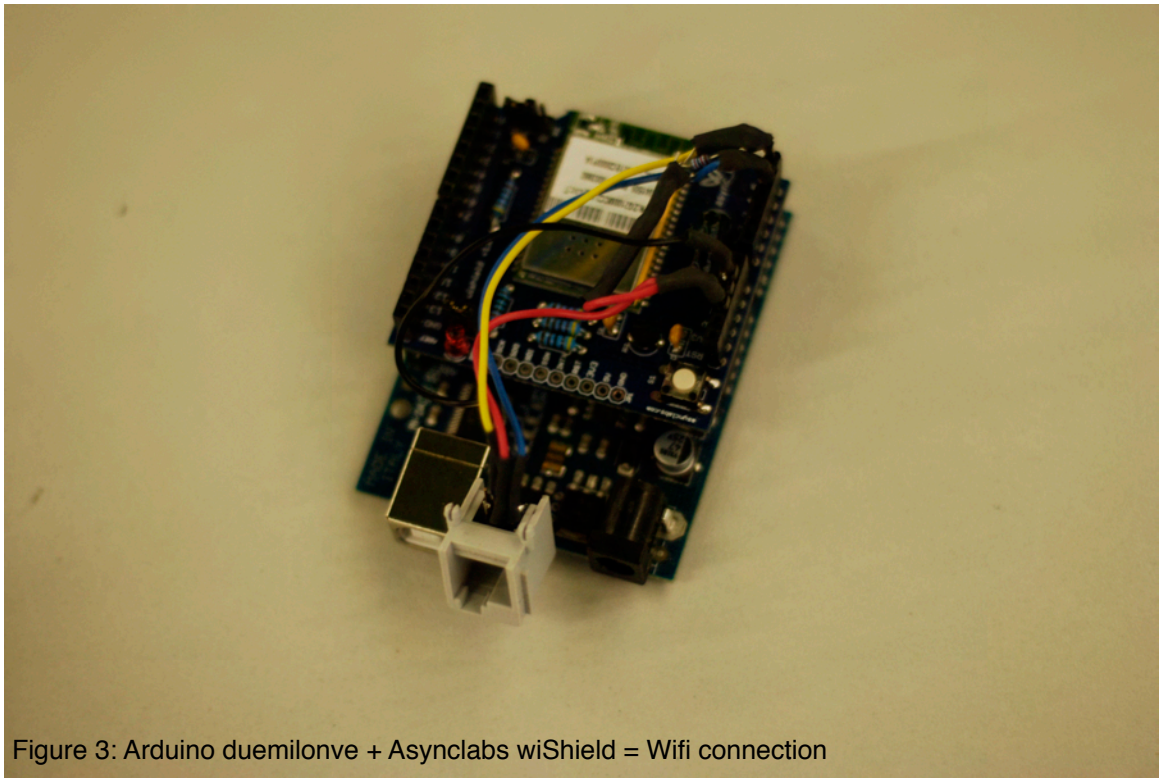


Figure 3: Arduino duemilonve + Asynclabs wiShield = Wifi connection

Circuit

The wiShield on top of the Arduino Duemilonove is a quite big package, therefore I wanted it to become smaller and designed my own circuit with the same wifi module as the wiShield uses. This module is ZG2100MC from ZeroG and because the wiShield is open source I could use part of the circuit of the wiShield to create a smaller product. Besides making the wiShield smaller I also changed the Arduino Duemilonova into an Arduino Mini Pro. This resulted into the schematic that can be found in figure 4. .

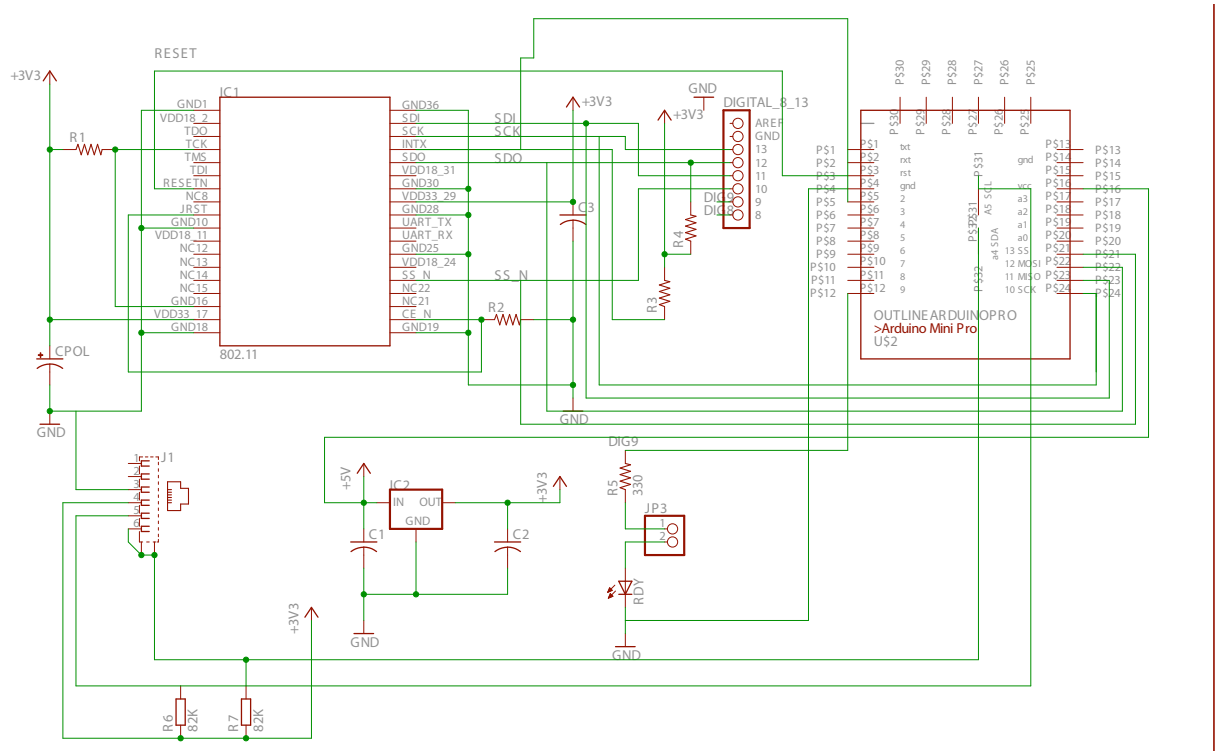


Figure 4: schematic WifiBlock

PCB

The ZG2100MC module has very small pads and I wanted to experiment with making my own printed circuit board I turned the schematic into a board drawing. I tried to make the PCB so small and effective as possible. Making a two sided print by hand is difficult, therefore I have chosen to use Air-wires for the Arduino instead. . The drawings of the PCB can be found in the figures 5-9.

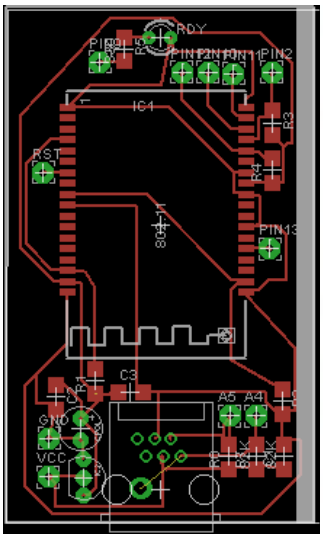


Figure 5 & 6: WifiBlock PCB drawing

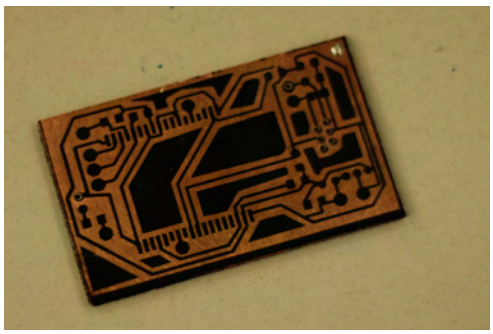


Figure 7: WifiBlock PCB before etching.

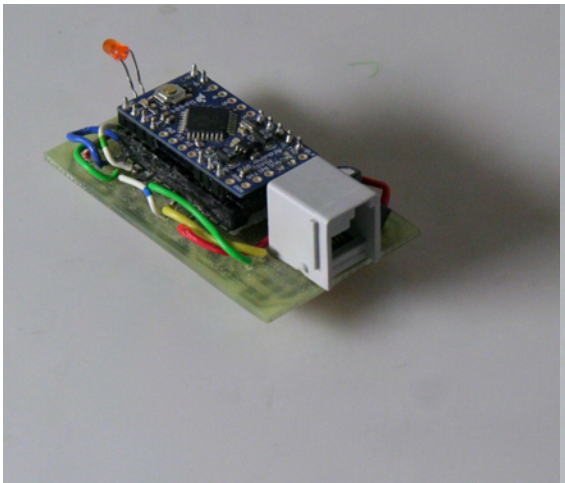


Figure 8: WifiBlock PCB top soldered

Figure 9: WifiBlock PCB bottom soldered

Casing

Next to the electronics it is also important to fit the wifi module to the lego environment. Therefore I have designed a lego brick casing with the following requirements:

- Use the already existing form language of lego sensors
- Use the same dimensions of connection points as lego
- The product should be easy to assemble.
- Easy to open and close.

With these requirements I have created different designs and my final design can be seen in the following pictures. I have used three colors to mimic the lego form language (white, grey, orange).

Figure 10 shows a picture of the final casing model.

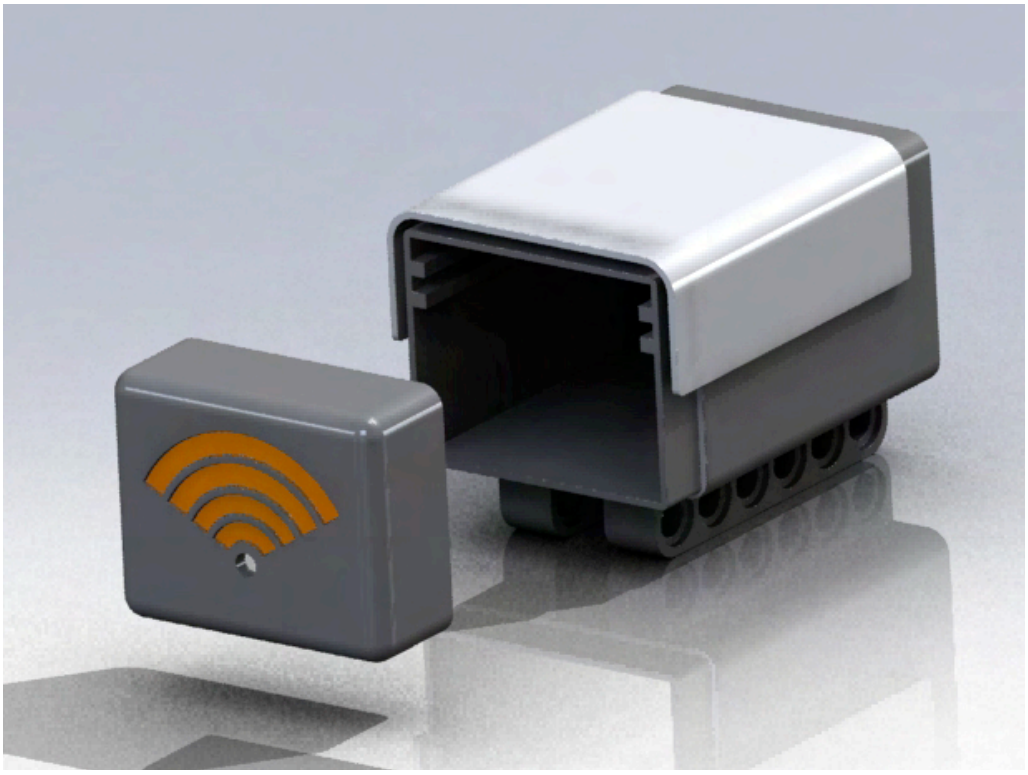


Figure 10: WifiBlock casing assembly

Figure 11 shows a technical drawing of the total casing assembly. More technical drawings of individual parts of the assembly can be found in the appendix A1.

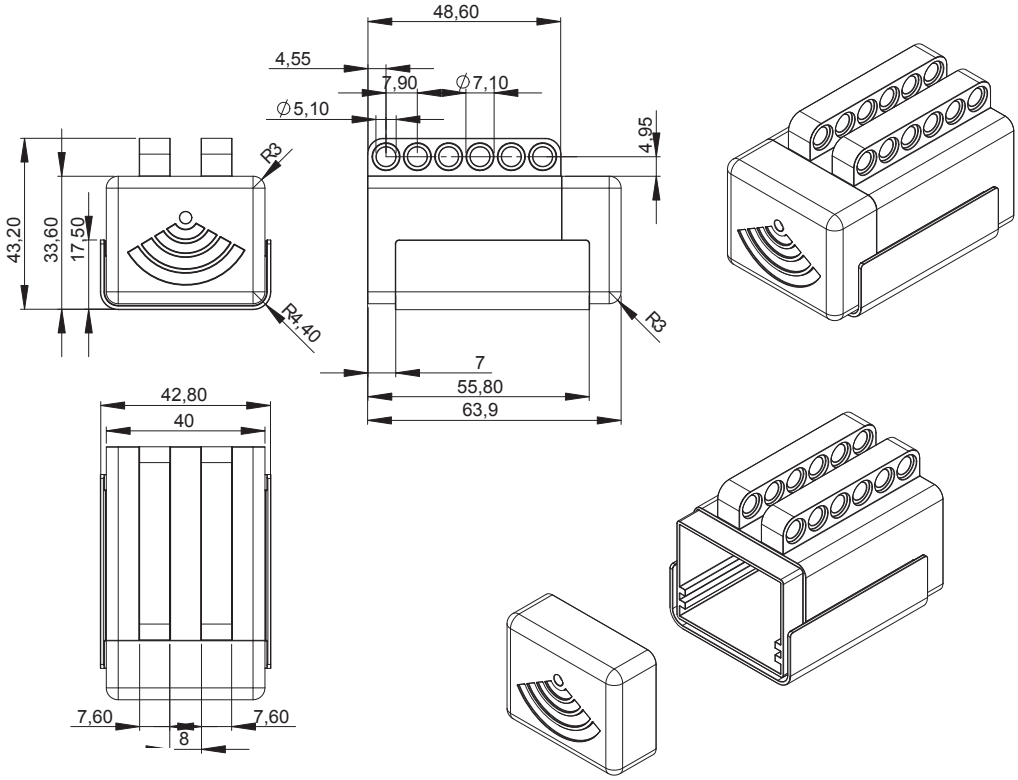


Figure 11: Technical drawing of WifiBlock casing assembly

Under the brick I have designed connection strips for lego. There are two strips positioned with the possibility to have one strip in between. Figure 12 shows the WifiBlock connected to ordinary lego pieces.

Figure 12: Pictures of the WifiBlock connected to ordinary lego pieces.

As can be seen in the figure 13 the electronics consist out of one part and can be slide into the brick easily. The PCB will be fixed by the connector hole.

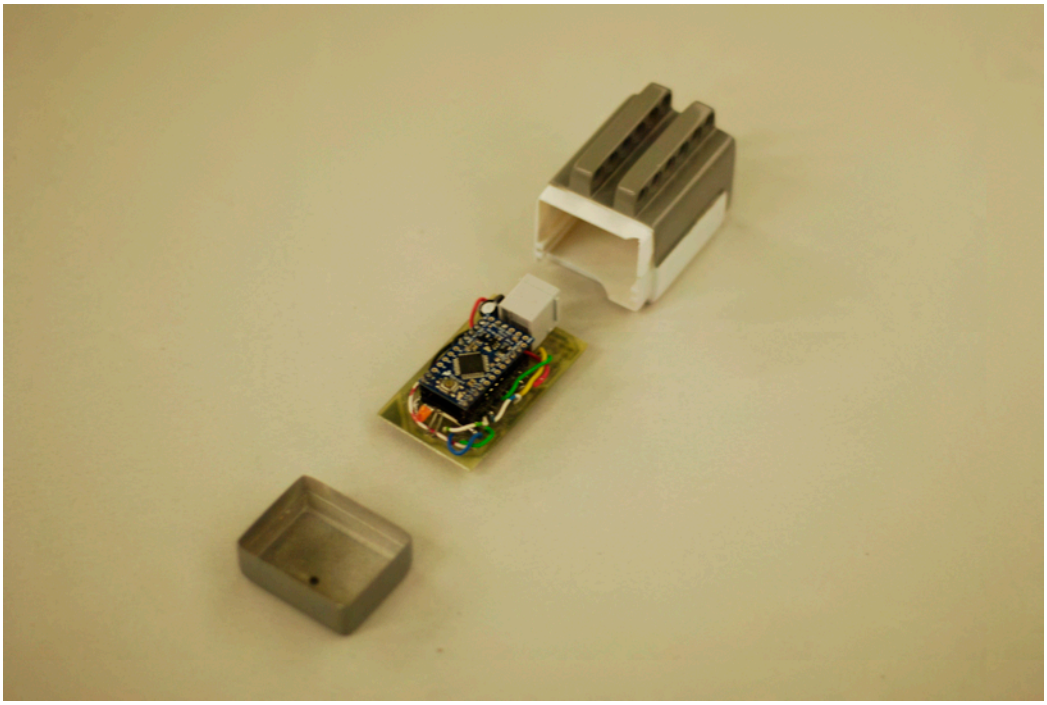


Figure 13: Exploded view of the WifiBlock

The orange led in the front indicates whether there is a connection to a wifi access point.

Figure 14: Led that indicates the WifiBlock has a connection to the internet

The Software

The software part of the WifiBlock consist out of three elements: connecting the Arduino to the NXT via I2C, connecting the Arduino to the wiShield via SPI and a protocol that converts the commands from the NXT to the wiShield. Although, I have learned a lot about internet connections and the I2C protocol the most difficult part was writing my own protocol to connect a Java Class to the wiShield library.

The following scheme gives the an overview of the different libraries and software packages that are used within the extension. I have written the parts in red.

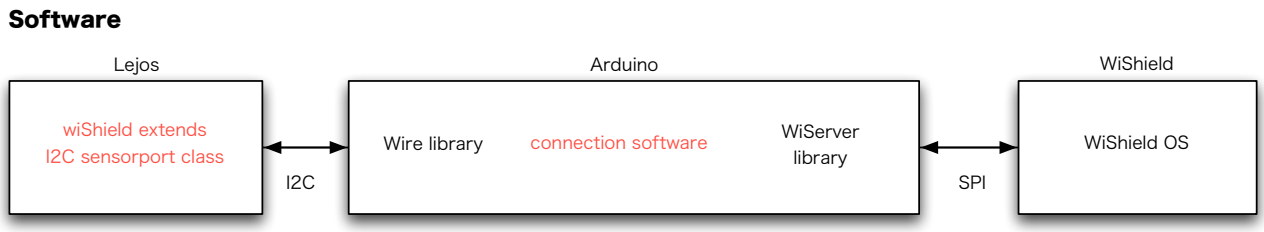


Figure 15: Software protocols of the WifiBlock

Making the I2C connection NXT --> <-- Arduino

The NXT has a build in I2C communication port and Lejos has a class which can be used to design your own I2CsensorPort. This class is necessary to create your own sensor and in my case connect to the Arduino. The Arduino has an I2C library called Wire. The Wire Library allowes the Arduino to be Slave or Master within an I2C protocol. Within all Lego NXT applications the NXT brick is the master and the rest are slaves.

The I2C bus has been developed in the beginning of the 80's by Philips with the intend to create an easy connection between processor and chips of a television. The advantage of this system is that it only uses two wires, which is much more efficient than the Byte Wide system which has been used until the introduction of I2C.

The two wired system is possible because the data is transmitted serially, one bit at a time. One wire is for sending and receiving data (SDA) and the other is providing a clock (SCL). Important to know is that an I2C bus always has one Master node - in our case the NXT - and can have up to 127 Slave nodes.

The transfer starts by sending a start byte (S), the Data line (SDA) is pulled low, while the Clock line (SCL) stays high. Then, SDA sets the transferred bit while SCL is low (blue) and the data is received when SCL rises (green). When the transfer is complete, a stop byte (P) is sent by pulling SDA up, while SCL also remains high.

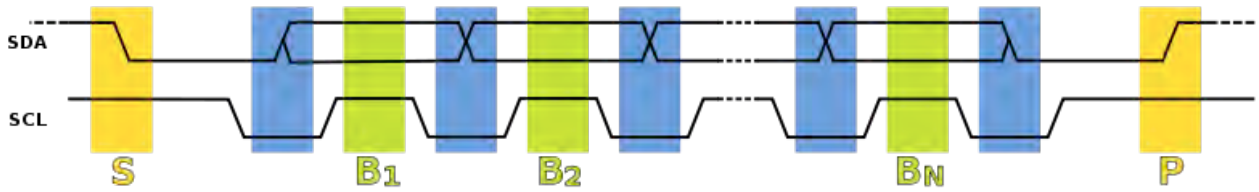


Figure 16: I2C protocol sequence. (wikipedia)

Although both Lejos and Arduino have an existing I2C library it turned out that timing still is very important within Arduino to get proper communication. The Arduino can have interrupts onSend

and onReceive, but when too much processing is done within these interrupts the timing is influenced and the communication is lost. In order to use it correctly the data processing should be not executed in the interrupts of the Arduino.

To implement the I2C communication in Lejos you need the following parts of code. First you need to import the I2Csensorport class, then you need to set the port of the NXT and the address of the communication and finely the data can be transmitted or requested. The following code snippets show how the implementation should be done.

```
I2CSensor Arduino = new I2CSensor(SensorPort.S2); //Define NXT Port of WifiBlock
byte[] buf = new byte[6]; // Buffer Array to send/store communication data
Arduino.setAddress(27); // The Address of the Slave || WifiBlock

for(int i = 0; i <6; i++){ // Write data into array
 buf[i] = 5;
}
Arduino.getData(0, buf, buf.length()); || Arduino.sendData(0,buf,buf.length()); //Send or get data
```

The full Lejos code about communication can be found in Appendix C1

To implement the I2C communication in Arduino two interrupts have to be initialized and the communication address has to be set similar to the address of the NXT. After the interrupts

```
void setup() {
  Wire.begin(27);                // join i2c bus with address 127
  Wire.onReceive(receiveEvent);
  Wire.onRequest(requestEvent);   // request event
}
```

functions are initialized in the setup function, the functions can be expanded.

```
void requestEvent()
{
    Wire.send(sender, senderLength); // sender = byte[] && senderLength = 6
  }

// ---------------------------------------------------------------------------
// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void receiveEvent(int howMany)
{
  char URL[howMany];
  int availables = howMany;
  while(0 < availables)  // loop through all but the last
  {
    availables = Wire.available();
    char c = Wire.receive(); // receive byte as a character
    URL[howMany-availables] = c;
  }

}
```

The full Arduino I2C communication can be found in Appendix C2.

Making the Wifi --><-- Arduino connection

Just as the I2C protocol is already implemented into an Arduino library the SPI library can be used as well. However, the producers of the wiShield made it even more simple, they have written their own library to control the wifi module without having to concern about the protocol. This allowed me to focus on the internet connection.

SPI is a protocol is originally designed by Motorola and is very similar to I2C only it is faster, full-duplex, and has no specific address for a slave. Instead of a 2 wired system it is a 4 wired system. To begin a communication, the master first configures the clock, using a frequency less than or equal to the maximum frequency the slave device supports. Such frequencies are commonly in the range of 1-70 MHz.

The master then pulls the slave select low for the desired chip. During each SPI clock cycle, a full duplex data transmission occurs:

- the master sends a bit on the MOSI line; the slave reads it from that same line
- the slave sends a bit on the MISO line; the master reads it from that same line

The implementation of the AsyncLabs WiShield Library comes with a lot of parameters that need to be set. First the IP address of the module, the Gateway address, the subnetmask, the SSID, the password mode, the password need to be set. After that the wiShield can be configured to host a page and and interrupts can be written to process data returned by a server.

```
// Wireless configuration parameters ----------------------------------------
unsigned char local_ip[] = {192,168,0,3};           // IP address of WiShield
unsigned char gateway_ip[] = {192,168,0,1};      // router or gateway IP address
unsigned char subnet_mask[] = {255,255,255,0}; // subnet mask for the local network
const prog_char ssid[] PROGMEM = {"xxxxxxxx"};          // SSID max 32 bytes

unsigned char security_type = 2;           // 0 - open; 1 - WEP; 2 - WPA; 3 - WPA2

// WPA/WPA2 passphrase
const prog_char security_passphrase[] PROGMEM = {"xxxxxx"};    // max 64 characters

// A post Requests for the wiki. :D

void setup() {

  // Initialize WiServer (we'll pass NULL for the page serving function since we don't need to serve web pages)
  WiServer.init(sendMyPage);// sendMyPage should be written as a function and declared somewhere else.

  // Enable Serial output and ask WiServer to generate log messages (optional)

  WiServer.enableVerboseMode(true);
  // Have the processData function called when data is returned by the server
  getVariables.setReturnFunc(printData); // Define interrupt function for data return.
}
```

By setting the home page function to sendMyPage, this function is called every time a client sens an URL request to the IP address of the wiShield. The webpage hosted by the wiShield can be designed by using the “wiServer.print” function. The WiServer.print does support normal html code. The following code example will display a button when the IP address of the wiShield is called.

```
WiServer.print("<html><head><title>Led switch</title></head>");
WiServer.print("<body>Controlling the robot <form method=GET>”);
WiServer.print(“<input type=submit name=DIR value=UP></form>");
WiServer.print("</body>");
WiServer.print("</html>");
```

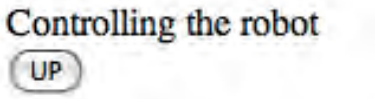


Figure 17: page generated by wiShield

The printData interrupt function will be called whenever an getURLrequest is executed by the arduino. The response of the server will be returned into the printData function, which receives the data into a character pointer and the length of the pointer array. The data can be parsed and used to your convenience. Here is an example code.

```
// Function that prints data from the server
void printData(char* data, int len) { //char* data = data returned by server && len = length of the data

  // Print the data returned by the server
  // Note that the data is not null-terminated, may be broken up into smaller packets, and
  // includes the HTTP header.
  while (len-- > 0) {
    Serial.print(*(data++));
  }
}

// IP Address for www.weather.gov
uint8 ip[] = {140,90,113,200};

// A request that gets the latest METAR weather data for LAX
GETrequest getWeather(ip, 80, "www.weather.gov", "/data/METAR/KLAX.1.txt"); // Declare een URL request with these
parameters. It will request (http://140.90.113.200/data/METAR/KLAX.1.txt)
```

By designing the information of a remote server carefully, you can make it easier to parse data in the wiShield. For example after “<n” the parsing should start to store the rest of the data into an array in the Arduino. The following example shows a PHP file that stores the getRequest variables into an database and returns the special “<n” characters to be the beginning of a parsing in the Arduino code.

The PHP code to store getRequest data into an database and return data.

```
<?php
$con = mysql_connect("localhost","root","root");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("test", $con);

$sql="INSERT INTO Omgeving (SPACE,ACTIE,AFSTAND) VALUES ('$_GET[v1]','$_GET[v3]','$_GET[v2]')";
if (!mysql_query($sql,$con))
  {
  die('Error: ' . mysql_error());

  }
echo "1 record added";
echo "<n hoi";  //Here is the special character for the beginning of the ParseData
mysql_close($con);

?>
```

The Arduino code to parse the data after the “<n” character.

```
// Function that prints data from the server
void printData(char* data, int len) {

  // Print the data returned by the server
  // Note that the data is not null-terminated, may be broken up into smaller packets, and
  // includes the HTTP header.
  int parsData = 0;
  int length = len;
  for(int i = 1; i <len; i++) {
      if(data[i] == 'n' && data[i-1] == '<') { //Determine the data signal “<n” and start parsing.
         parsData = i-1;
         i = len;
      }
  }
 //Specific data that is necessary for the NXT.
  for(int i = parsData; i < (parsData + 28) && i < length; i++) {
    dataBack[i] = data[i];
    Serial.print(dataBack[i]);
  }
```

It is also possible to send variables within the getURLrequest, using the normal HTML protocol. So instead of calling the URL “/data/METAR/KLAX1.txt” you can also send variables along with the request by using the following syntax. “/somefile.php?var1=value1&var2=value2” etc. To be able to use this variables the somefile.php should turn the variables into something useful, for example store them into an database.

Extension Class design a protocol

Now that I have connected the Arduino both to the wifi module and the NXT, I have to design a protocol that makes it possible to send data from the NXT to the wiShield. Therefore I created a protocol with buffers and arrays. Although it would be nice if the WifiBlock could be fully programmed in Lejos, creating such java class is complicated and I think it is more efficient to have control over the Arduino software as well as the NXT software. Being able to program both opens up the possibilities and enables the wiShield to be used in more diverse ways for advanced users

To support new users in the use of the WifiBlock I designed a java class which can be used to send and receive information easily from the WifiBlock. To smoothen this process I also have written basic Arduino NXT software (WiFiBlock Arduino) which enables the wiShield to be Server and Client simultaneously.

The combination WifiBlock Arduino Software and the WifiBlock Class enables to user to:

- set the addresses, gateway, passwords and all other important wifi connection parameters within Lejos.
- Have a webpage hosted by the WifiBlock with controlbuttons to control the NXT over the internet.
- send a getURLrequest to any IP address in combination with variables to store data into a database
- get the URLRequest data back to the NXT

With these basic features you should be able to write simple applications that can be used in many ways. However if more dedicated actions are required it is advisable to adapt the WifiBlock Arduino software and write a dedicated solution.

The WifiBlock Class

The WifiBlock Class is a class that can be used to control the WifiBlock within Lejos. I divided the class into two different parts: sending and receiving data. First I will start with the sending protocol. This protocol is based upon the principle of writing to a specific register in the I2C communication. When sending data from the NXT towards the WifiBlock different register can be used to easily parse the data in the Arduino.

The I2Csensor class does support the writing to different registers. Within the sendData and getData functions the first parameter is the register the NXT is writing to. The second parameter is a byte array which stores the actual data and the third parameter is the length of the byte array.

```
// the byte array which will be used to save the data
byte [ ] readData = new byte [ 1 1 ] ;
getData ( 0 , readData , readData.length()) ;
```

For writing towards the WifiBlock the different registers seem to work fluently. And I have chosen to allocate different messages to different registers. This is the register protocol I have designed to send data from the NXT to the Arduino:

```
/* The protocol of the NXT towards WifiBlock.
            *  0 = WifiBlock init(), asks the wifiblock to make a connection
            *  1 = local_ip of WifiBlock; sets the ip address
            *  2 = gateway of WifiBlock; sets the gateway address
 *  3 =            subnet of WifiBlock; sets the subnet mask
 *  4 =            ssid of Wifiblock; sets the SSID of the WifiConnection
 *  5 = security mode of the            WifiBlock. (0 = open, 1 = WPA, 2 = wpa2)
 *  6 = password of wireless network.
 *
            *  19 = set new IP address for an URL request.
            *  20 = set a new URL for getRequest WifiBlock. Linked to a php script that does something with it.
 */
```

Here is an example of the implementation of code within Lejos and code within the onRequest() function in Arduino to give an impression how it works. The full Arduino Code and WifiBlock class can be found in the appendix D1 and D2.

Part of the Arduino code: receiveEvent

```
 void receiveEvent(int howMany)
{
   char c;
  int s;
  int availables = howMany-1;

 //The first char should be the register byte. I set this to 20 in the java code
            if(Wire.available()) {
                        s = Wire.receive();
      }
 switch(s) {
             case 20:
                        while(0 < availables) // 20 is indicated in the send Register.
                        //loop through all but the last Check if for valueable address
                        {
                                    availables = Wire.available();
                                    char c = Wire.receive(); // receive byte as a character
                              URL[(howMany-1)-availables] = c;
                        }
                              getFullValue = URL;
                              submitVal = true;
             break;
            }
}
```

Part of the WifiBlock class

```
/* Function to send a URL or string to wifiBlock */
            public boolean sendToWifiBlock(byte[] URL){
             if(URL.length <= 32) {
                              this.sendData(20,URL, URL.length); // 20 is register to write an URL
                        return true;
             } else {
                        return false;
             }
}
wiShield.sendToWiShield("/lib.php?v1=A&v2=0000023&v3=23");
```

For the getData() out Lejos it is more difficult to request for data with different identifiers. The Arduino software does not support writing to different registers so it will write always to the same register. Therefore, I choose to use the first byte of the array that is send towards the NXT as identifier byte. Within the WifiBlock Class and the Arduino code I use the following protocol to get data from the WifiBlock. :

```
/* The protocol to get data from the arduino/WifiBlock to the NXT
            * The first byte of the Byte[] received data holds the kind of message.
 *
 * data[0] == "!" || 33 gives no new data.
 * data[0] == "#" || 35 gives data from the internet
 * data[0] == "%" || 37 gives commands for the robot.
            * data[0] == "&" || 38 the WiShield is active
 *
 */
```

One important constraint for the communication between the Arduino and the NXT is that the communication can only consist out of 32 bytes per data transfer. Therefore every dataRequest from the NXT towards the Arduino 32 bytes long. The first byte identifies the content the the rest of the bytes contain.

Within the current implementation of the WifiBlock source code, I only use one message that is send towards the NXT: the variables to control the robot via the internet. Although I also made it

possible to send server responds towards the NXT I did not implement the processing of it in the WifiBlock class yet.

So within the basic NXT Arduino code I define one array of 32 bytes and every time the NXT asks for data it receives this array. Depending upon code of the Arduino this array can have multiple meanings. This example shows how the first byte changes after a request has been send by the NXT.

```
byte dataBack[32] = "/library.php?v1=A&v2=348&v3=Re"; // Variable to send data back.
//Function to give values back to NXT
void requestEvent()
{
    Wire.send(dataBack, 32); // The data array that is send is always the same. Within the Arduino code it should change.
 dataBack[0] = 33; //reset the first byte so it will now there is no new data.
}
```

Functions and attributes of WifiBlock class

Next to the basic protocol that is explained in the previous part the WifiBlock Class has multiple functions to make it easier to integrate the WifiBlock into an application.

The constructor of the WifiBlock needs 7 parameters to be operate. These parameters consist out of the basic information of the Wifi connection. The constructor and the parameters can be seen in the following block:

```
public WifiBlock(I2CPort port, byte[] local_ip, byte[] gateway, byte[] subnet, byte[] ssid, byte security, byte[] password) {
                        super(port);
                        super.setAddress(27); // The I2Caddress is default 27 corresponding with Arduino
                        this.setIP(local_ip);
                        this.setRouterIP(gateway);
                        this.setSubnetMask(subnet);
                        this.setSSID(ssid);
                        this.setProtectionMode(security);
                        this.setPassword(password);
}
```

Next to the constructor function there are other functions and global variables that help to implement the WifiBlock. The following block shows the header file of all functions. The total

```
/*Global variables that give information about the wifiBlock   */
static byte[] dataReceiveBuf = new byte[32]; // databuffer for the checkDataAvailable function of WifiBlock
static boolean NewDataAvailable = false;  //Is there new data available?
static int statusOfDataBack = 0;  //What kind of data is received from the WifiBlock
static boolean isActive = false;  //Status of the connection

/* Functions that help to start and play with the WifiBlock */
public void startWiShield() ; //Set the WifiBlock to make a connection
public boolean sendToWiShield(byte[] URL); /* Function to send string to wiShield, returns boolean if transfer is succes */
public void setIPGetRequest (byte[] ipAddress); /*Set the ip Address for a new GetRequest*/
public void setPortGetRequest(int port); /* Set the Port number for a new GetRequest */
public byte[] checkDataAvailable(); /*Function that gets data out of the WifiBlock sets NewDaaAvailable to true*/
public void setIP (byte[] ipAddress); /*Set the ip Address of the WifiBlock*/
public void setRouterIP (byte[] routerAddress): //Set the Routers IP Addres where the wifi module has to connect to /
public void setSubnetMask (byte[] subnet); /*Set the Subnet of the wireless connection */
public void setSSID (byte[] ssid); /*Set the SSID of the wireless connection */
public void setProtectionMode(byte secMod): // Set the protection mode of router 0 = open; 1 = WPA; 2 = WPA2//
public void setPassword (byte[] password); /*Set the Password for the wireless connection */
public boolean isActive() ;    //Returns the state of the WifiBlock. If connected then true otherwise false.
public boolean newDataAvailable(): //Returns true if there is newDataAvailable
```

code can be found in appendix D1.

Application examples

The extension class for the NXT and WifiBlock is a real building block and it enables the user to do whatever they intend to do with the internet. To give a better understanding of the extension class and the potential of the WifiBlock in general I have written two different applications. One in which the NXT can be controlled via the internet and the other in which the NXT will explore the room and export his sensor data into a database. Which can be visualized with a flash file.

Controlling the NXT via the internet

There are two possibilities to use the WifiBlock over the internet. The WifiBlock is a Client or it is a server. Although both options can work I have chosen to use the WifiBlock as server in this example.

The WifiBlock is able to host a website at his IP address and to be able to have buttons for control on the website I used the form getfunctions of html “<form method=GET>” to give data within the URL. Everytime the IP address of the WifiBlock is called it will generate the webpage of figure 18:

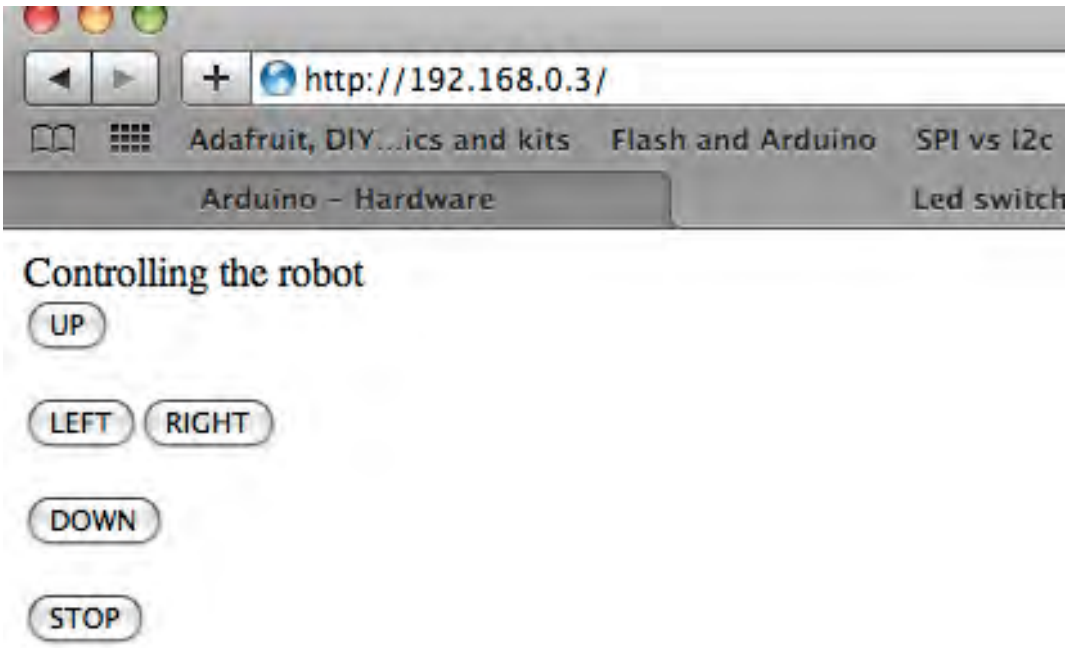


Figure 18: page generated by WifiBlock

By pushing a button the URL changes from for example http://192.168.0.3/ to http://192.168.0.4/?DIR=STOP. This change can be processed by the WifiBlock and translated into different variables in the dataBack array. That data can be read by the NXT. Down here is the important part of the Arduino code where the webpage is generated and the URL info is processed. The total arduino code for this example can be found in appendix D2.

```
// This is our page serving function that generates web pages
boolean sendMyPage(char* URL) {

 // Serial.println("Page printing begun");
  //Serial.println(URL);

  //check whether we need to change the led state
  if (URL[1] == '?' && URL[2] == 'D' && URL[3] == 'I' && URL[4] == 'R') //url has a leading /
  {
    clearSenderArray();
   // Serial.println(URL[6]);
    switch(URL[6]) {
      case 'U':
            dataBack[0] = 255;
            break;
      case 'L':
            dataBack[1] = 255;
            break;
      case 'R':
            dataBack[2] = 255;
            break;
      case 'D':
            dataBack[3] = 255;
            break;
      case 'S':
            dataBack[4] =255;
            break;
    }
    dataBack[5] = 255;

    //after having change state, return the user to the index page.
     WiServer.print("<html><head><title>Led switch</title></head>");
     WiServer.print("<body>Controlling the robot <form method=GET><input type=submit name=DIR value=UP>");
     WiServer.print("</form><form method=GET><input type=submit name=DIR value=LEFT>");
     WiServer.print("<input type=submit name=DIR value=RIGHT></form>");
      WiServer.print("<form method=GET><input type=submit name=DIR value=DOWN></form>");
     WiServer.print("<form method=GET><input type=submit name=DIR value=STOP></form>");
      WiServer.print("</body>");
     WiServer.print("</html> ");
    return true;
  }
```

For this example the only thing I need to do in Lejos is constantly asking for data and see whether it changes. Depending on the variables of the data the robot could go into different states. This lejos code has no full implementation of the WifiBlock Class. Within this example I have chosen to define the basic parameter within the Arduino code, which allows me forget about the Wifi connection parameters.

```
import lejos.nxt.Button;
import lejos.nxt.LCD;
import lejos.nxt.Motor;
public class controlRobotv2 {

        // Wireless configuration parameters ----------------------------------------
        static byte local_ip[] = {(byte) 192,(byte) 168,(byte) 0,(byte) 4};      // IP address of WiShield
        static byte gateway_ip[] = {(byte) 192,(byte) 168,(byte) 0,(byte) 1}; // IP address of WiShield
        static byte subnet_mask[] = {(byte) 255,(byte) 255,(byte) 255,(byte) 0};         // IP address of WiShield

        public static void main(String[] args) {
                WifiBlock Arduino = new WifiBlock(SensorPort.S2);
         byte[] buf = new byte[32];
         LCD.refresh();
                Motor.A.setSpeed(800);// 2 RPM
                Motor.B.setSpeed(800);// 2 RPM

                Arduino.startWiShield();

                while(!Button.LEFT.isPressed()){
                        buf = Arduino.getDataWiShield();

                 for(int i = 0; i < 6; i++){
                        LCD.drawInt(i, 0, i);
                        LCD.drawInt((int) buf[i], 2, i);
                 }

                 if((int) buf[4] == -1) {
                                Motor.A.stop();
                                Motor.B.stop();
                 } else {
                        if((int) buf[0]==-1) {
                                        Motor.A.forward();
                                        Motor.B.forward();
                        }
                        if((int) buf[1]==-1) {
                                        Motor.B.rotate(180,true);
                                        Motor.A.rotate(-180, true);
                        }
                        if((int) buf[2]==-1 ){
                                        Motor.A.rotate(180,true);
                                        Motor.B.rotate(-180,true);
                        }
                        if((int) buf[3]==-1) {
                                        Motor.A.backward();
                                        Motor.B.backward();
                        }
                 }
         }
        }
}
```

Exploring the room

After the fairly simple example of controlling the robot via the internet I also designed an application in which the robot is driving around an unknown room and where it is constantly driving and colliding. Every time it changes action (from drive to turn and visa versa) it sends the amount of action into a database. Eventually when the robot has driven long enough, this data can be turned into a map of the room.

This example combines five different stages into one product which makes it possible to see the route of the robot in a window on the computer. To be able to do so the data flow is as follows:

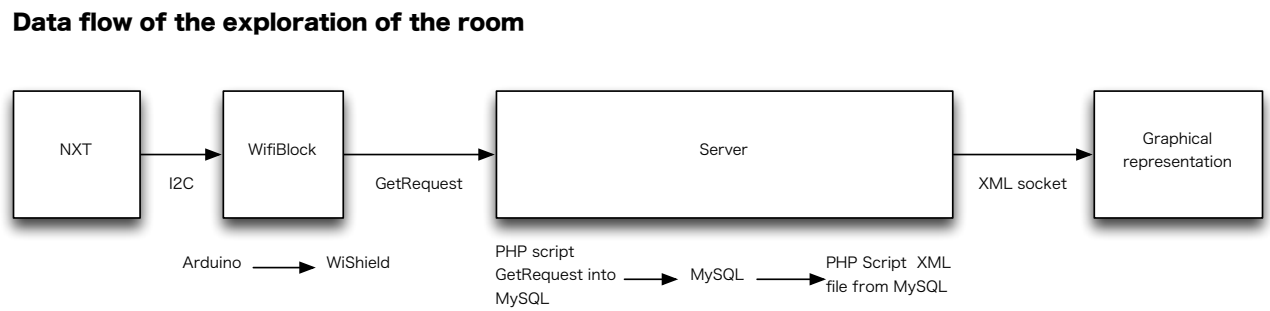


Figure 19: DataFlow of the exploring room example.

The NXT software

The only thing the NXT software has to do is drive forward, wait until one sensors detects an object send the traveled distance a an URL formatted like : "/lib.php?v1=B&v2=0000234&v3=Vo". In this URL the v1 is the room, v2 is the traveled distance and the v3 is the action corresponding to the travel. Then the robot should turn and send information about the angle in a corresponding format to the URL.

The URL data will be translated by the Arduino into a getRequest. The total lejos Code can be found in Appendix E1.

The WifiBlock software

The software for the WifiBlock is the basic WifiBlock source code. Within the software there is a function onRequest and if data is written to register 20 a new Get URLrequest will be executed as soon possible. The source code of the WifiBlock can be found in Appendix E2.

The PHP script to translate the getURLrequest into MySQL

The getURL request of the WifiBlock is for example follows: http://192.168.0.100//lib.php?v1=B&v2=0000234&v3=Vo. The three different parameters can be translated into variables by using the “$_GET[v1]” command in PHP. It is also easy to connect to a database with PHP and by combining these two the following PHP script will process the variables and store them into a table.

```
<?php
$con = mysql_connect("localhost","root","root");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
mysql_select_db("test", $con);
$var4 = date("Y-m-d H:i:s");
echo $var4;
//$sql="INSERT INTO Omgeving (SPACE, TIJD, ACTIE, AFSTAND)VALUES ('$_GET[var1]',date("H:i:s m-d-
Y"),'$_GET[var2]','$_GET[var3]')";
$sql="INSERT INTO Omgeving (SPACE,ACTIE,AFSTAND) VALUES ('$_GET[v1]','$_GET[v3]','$_GET[v2]')";
if (!mysql_query($sql,$con))
  {
  die('Error: ' . mysql_error());
    }
echo "1 record added";
echo "<n hoi";
mysql_close($con);
?>
```

Visualize the route of the robot

After the variables are stored into the database you can see them by using PHPmyAdmin (see picture) for example, but you can also read the database and visualize the routing of the robot.

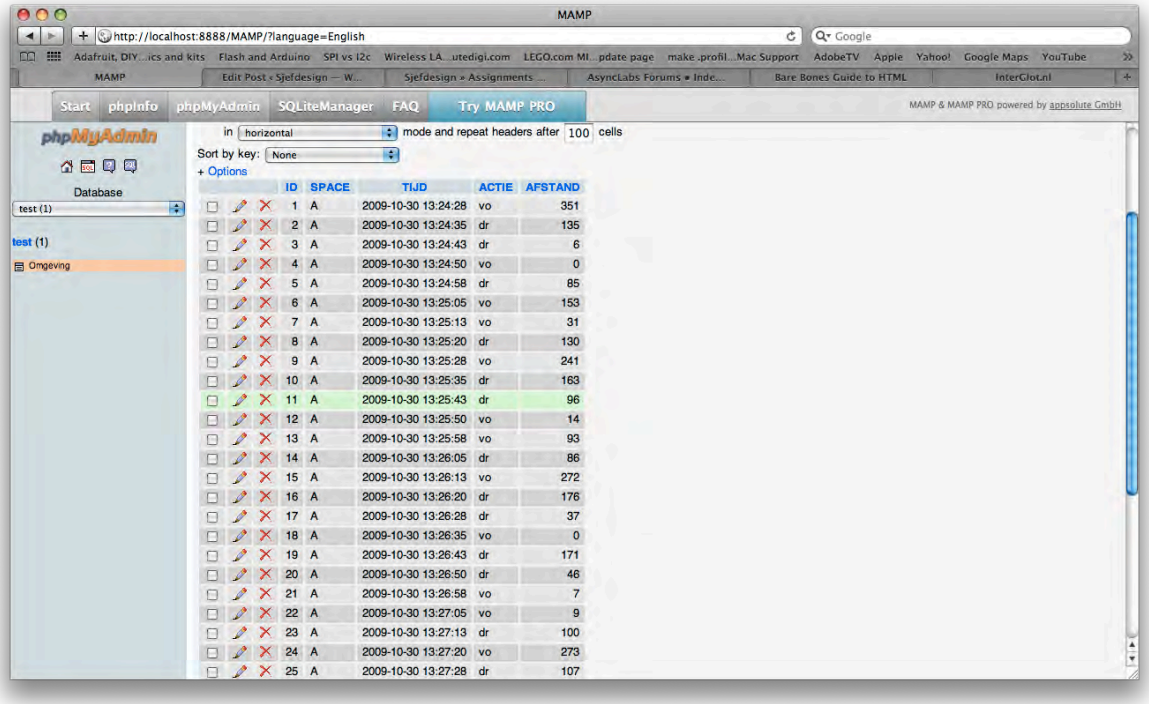


Figure 20:PHPmyAdmin panel to see the data gathered by the NXT into a database

I used flash with an XML socket to visualize the routing of the robot. However Flash is not able to read an MySQL database and therefore another PHP script should translate the database file into an XML file that can be read by Flash.

The script of the PHP file is the following:

```
<?php

$con = mysql_connect("localhost","root","root");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
mysql_select_db("test", $con);
$result = mysql_query("SELECT * FROM Omgeving WHERE SPACE='A'");

//now we are going to write this data into xml for flash.

echo("<?xml version=\"1.0\" encoding=\"iso-8859-1\"?>\n");
// front slash to skip inverted commas.

echo("<products>\n");
while($rows=mysql_fetch_assoc($result)){

echo("<item><actie>" . $rows['ACTIE'] ."</actie>");
echo("<afstand>" . $rows['AFSTAND'] . "</afstand>\n");
echo("</item>");
 }
echo("</products>");

?>
```

The XML that is returned by this script can be used in Flash directly. The following code shows a small part of the existing XML file.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<products>
<item><actie>vo</actie><afstand>351</afstand>
</item><item><actie>dr</actie><afstand>135</afstand>
</item><item><actie>dr</actie><afstand>6</afstand>
</item><item><actie>vo</actie><afstand>0</afstand>
</item><item><actie>dr</actie><afstand>85</afstand>
</item><item><actie>vo</actie><afstand>153</afstand>
</products>
```

With this XML file and the following ActionScript 3 code the visualizations as in the figure 21 can be made.

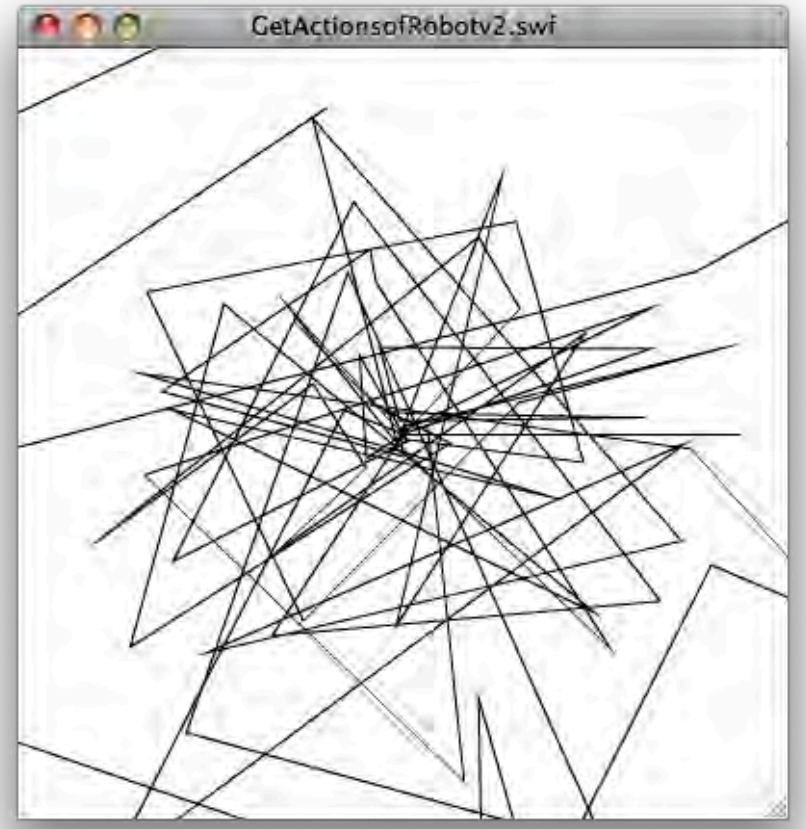
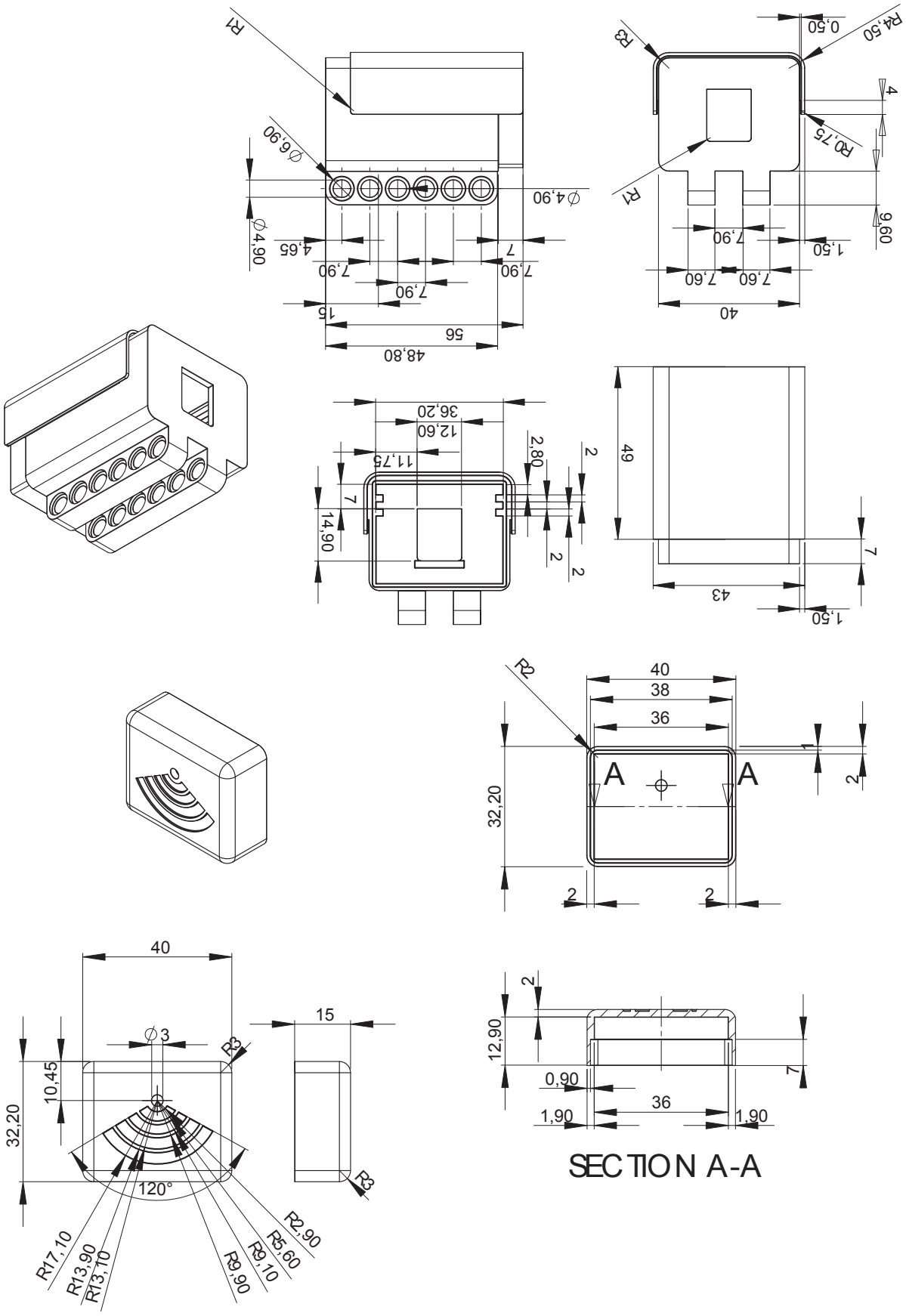


Figure 21: drawing of the room generated by flash.

```
import flash.display.*;

var xmlLoader:URLLoader = new URLLoader();
var xmlData:XML = new XML();
xmlLoader.addEventListener(Event.COMPLETE, LoadXML);
xmlLoader.load(new URLRequest("http://localhost:8888/code.php"));//code.php is de echte

//Interval Timer to reload the xml file sometimes.
var intervalDuration:Number = 10000;// duration between intervals, in milliseconds
var intervalId:uint = setInterval(myRepeatingFunction, intervalDuration);

var lineDraw:Shape = new Shape();
lineDraw.graphics.lineStyle(1, 0x000000, 1, false, LineScaleMode.VERTICAL,
                        CapsStyle.NONE, JointStyle.MITER, 10);
//lineDraw.graphics.moveTo(400,400);
var amountConnections:int = 0; // The total amount of new movieclips

//var linesClip[amountConnection]:MovieClip = new MovieClip();

function LoadXML(e:Event):void {
        xmlData = new XML(e.target.data);
 ParseBooks(xmlData);
}
function ParseBooks(bookInput:XML):void {
 var newMovie:MovieClip = new MovieClip();
 newMovie.x = 200;
 newMovie.y = 200;
 var tempX = 0;
        var tempY = 0;
 var hoek = 0;

 var authorList:XMLList = bookInput.item;
 trace("Ik bedel maar " + bookInput.item.length());

        for each (var item:XML in bookInput.item) {
         if (item.actie.text() == "dr") {
                hoek = hoek + ((item.afstand /180) * Math.PI);
                //hoek = Math.cos(hoek);
                trace(item.afstand);
                trace("Hoek cos= " + Math.cos(hoek) + ". Hoek Sin=" + Math.sin(hoek));
         } else {
                tempX = (item.afstand/2) * Math.cos(hoek);
                        tempY = (item.afstand/2) * Math.sin(hoek);
                //trace(tempX + " : " + tempY);
                        lineDraw.graphics.lineTo(tempX, tempY);
                newMovie.addChild(lineDraw);
         }
        }
 addChild(newMovie);
}
function myRepeatingFunction():void {
 drawPrevChildsRed();
 amountConnections++;
        xmlLoader.load(new URLRequest("http://localhost:8888/code.php"));//code.php is de echte
}
```

Appendix A1

The technical drawings of the two casing parts.



Appendix C1

The full Lejos code to have I2C communication with the Arduino.

```
import lejos.nxt.*;
public class i2CCommunication {

        static String URL = "/library.php?v1=A&v2=200&v3=Re";//private static final int LEGO_MODE = 0;

        public static void main(String[] args) {

                I2CSensor Arduino = new I2CSensor(SensorPort.S2);
        byte[] buf = new byte[6];
                byte[] send = new byte[URL.length()];
        for(int i = 0; i <send.length; i++){
                 send[i] = (byte) URL.charAt(i);
        }
        for(int i = 0; i <6; i++){
                 buf[i] = 5;
        }
        LCD.refresh();
        LCD.drawInt(URL.length(), 6, 0);

        Arduino.setAddress(27);
                while(!Button.LEFT.isPressed()){

                 if(Button.ESCAPE.isPressed()){
                        Arduino.getData(0, buf, 6);
                        LCD.drawString("Bezig met Lezen", 0, 7);
                        try {
                                Thread.sleep(200);
                        } catch (Exception e) {
                        }
                        LCD.drawString("Klaar met Lezen", 0, 7);

                 }
                 for(int i = 0; i < 6; i++){
                        LCD.drawInt(i-5, 0, i);
                        LCD.drawInt((int) buf[i], 2, i);
                 }
                        if(Button.RIGHT.isPressed()) {

                        Arduino.sendData(0, send, send.length);
                        LCD.drawString("Bezig met Schrijven", 0, 7);
                        try {
                                Thread.sleep(200);
                        } catch (Exception e) {
                        }
                        LCD.drawString("Klaar met Schrijven", 0, 7);

                 }

        }

        }

}
```

Appendix C2

The full Arduino I2C communication with the NXT.

```
*Sketch to check the i2C connection between ARduino and NXT */
#include <string.h>
#include <Wire.h>   //Voor i2c communicatie

#define senderLength 6
// End of wireless configuration parameters ----------------------------------------

boolean states[3]; //holds led states
char stateCounter; //used as a temporary variable
char tmpStrCat[64]; //used in processing the web page
char stateBuff[4]; //used in text processing around boolToString()
char numAsCharBuff[2];
char ledChange;
//static int senderLength = 5;  // DE lengte van de sender byte buf.
byte sender[senderLength]; //Array voor de i2c sender.


void clearSenderArray(void) {
      for(int i = 0; i < senderLength; i++) {
     sender[i] = 0; //Putting everything of
    }
}

void requestEvent()
{
    //printer();
    Wire.send(sender, senderLength);
    sender[5] = 0;

}
// ---------------------------------------------------------------------------
// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void receiveEvent(int howMany)
{
  char URL[howMany];
  int availables = howMany;
  while(0 < availables) // loop through all but the last
  {
    availables = Wire.available();
    char c = Wire.receive(); // receive byte as a character
    URL[howMany-availables] = c;
  }

}

void setup() {
  // Initialize WiServer and have it use the sendMyPage function to serve pages
  Serial.begin(57600);
  Wire.begin(27);                // join i2c bus with address 127
  Wire.onReceive(receiveEvent);
  Wire.onRequest(requestEvent);   // request event
  clearSenderArray();
}

void loop(){
  // Run WiServer
  delay(100);
}
```

Appendix D1

The full WifiBlock class code

```
import lejos.nxt.I2CPort;
import lejos.nxt.I2CSensor;

public class WifiBlock extends I2CSensor {
		static byte[] dataReceiveBuf = new byte[32]; // databuffer for the getDataOut of wiShield
		static boolean NewDataAvailable = false;
		static int statusOfDataBack = 0;
		static boolean isActive = false;

		public WifiBlock(I2CPort port, byte[] local_ip, byte[] gateway, byte[] subnet, byte[] ssid, byte security, byte[]
password) {
			super(port);
			super.setAddress(27); // The I2Caddress is default 27 corresponding with Arduino
			this.setIP(local_ip);
			this.setRouterIP(gateway);
			this.setSubnetMask(subnet);
			this.setSSID(ssid);
			this.setProtectionMode(security);
			this.setPassword(password);
		}
		/* The protocol to send data to the arduino from the NXT
		 * This protocol is based upon writing to registers in the Arduino.
		 * Within the protocol the first byte that is send can be processed by Arduino
 *  0 = wiShield		init();
 *  1 = local_ip of wiShield;
 *  2 = gateway of wiShield;
 *  3 =		subnet of wiShield;
 *  4 =		ssid of wiShield;
 *  5 = security mode of the		wishield. (0 = open, 1 = WPA, 2 = wpa2)
 *  6 = password of wireless network.
 *
 *  18 = set new		portnumber for getRequest
		 *  19 = set new IP address for an getRequest.
		 *  20 = set a new end URL for getRequest wiShield. Linked to a php script that does something with it.
 *
		 *  GetRequest(IPaddress, PortNumber, nameRequest, end tag) example:
		 *  GETrequest getVariables(192.168.0.1, 8888, "localhost", /library.php?v1=A&v2=348&v3=Re);
 */

		/* The protocol to get data from the arduino/WifiBlock to the NXT
		 * The first byte of the Byte[] received data holds the kind of message.
 *
 * data[0] == "!" || 33 gives no new data.
 * data[0] == "#" || 35 gives data from the		internet
 * data[0] == "%" || 37 gives commands for the robot.
		 * data[0] == "&" || 38 the WiShield is active
 */
		public void startWiShield() {
			this.sendData(0, (byte) 1);  // Send 1 to the 0 register to activate WifiBlock
		}
		/* Function to send a URL or string to wiShield */
		public boolean sendToWiShield(byte[] URL){
			if(URL.length <= 32) {
			this.sendData(20,URL, URL.length);
				return true;
		 }		else {
				return false;
		 }
		}
```

```
/*Set the ip Address for a new GetRequest*/
public void setIPGetRequest (byte[] ipAddress) {
        this.sendData(19, ipAddress, ipAddress.length);
}

/* Set the Port number for a new GetRequest */
public void setPortGetRequest(int port){
        this.sendData(18, (byte) port);
}
/*Function to get data out of the wiShield */
public byte[] checkDataAvailable(){
        byte[] getData = new byte[32];  //32 is de waarde die ingesteld is in de arduino per definitie om terug te
sturen.
this.getData(10,getData, getData.length);
dataReceiveBuf = getData;
   return getData;
}
/*Function to get data out of the wiShield */
public byte[] getDataWiShield(){
        byte[] getData = new byte[32];  //32 is de waarde die ingesteld is in de arduino per definitie om terug te
sturen.
this.getData(0,getData, getData.length);
dataReceiveBuf = getData;
switch(getData[0]){
case 33:
        NewDataAvailable = false;    // data[0] == "!" || 33 gives no new data.
        break;
case 35:
        statusOfDataBack = 1; // data[0] == "#" || 35 gives data from the internet
                NewDataAvailable = true;
        break;
case 37:
        statusOfDataBack = 2; // data[0] == "%" || 37 gives commands for the robot.
        NewDataAvailable = true;
        break;
case 38:
        isActive = true;  //  data[0] == "&" || 38 the WiShield is active
        NewDataAvailable = false;
        break;
  }
   return getData;
}
/*Set the ip Address of the Wifi Module*/
public void setIP (byte[] ipAddress) {
        this.sendData(1, ipAddress, ipAddress.length);

}
/*Set the Routers IP Addres where the wifi module has to connect to */
public void setRouterIP (byte[] routerAddress) {
                this.sendData(2, routerAddress, routerAddress.length);//2 First byte to notify Arduino it is
Router IP
}
/*Set the Subnet of the wireless connection */
public void setSubnetMask (byte[] subnet) {
        this.sendData(3, subnet, subnet.length);
}
```

```
	public void setSSID (byte[] ssid) {
		this.sendData(4, ssid, ssid.length);
	}
	/*Set the SecMode of the wireless connection */
	public void setProtectionMode(byte secMod) {
		this.sendData(5, secMod); //Secmod[0] = 0 --> OPen; =1 --> WPA;=2 --> WPA2
	}

	/*Set the Password of the wireless connection */
	public void setPassword (byte[] password) {
		this.sendData(6, password, password.length);
	}
	//Returns the state of the WifiBlock
	public boolean isActive() {
		return isActive;
	}
	//Returns if there is newDataAvailable
	public boolean newDataAvailable() {
		return NewDataAvailable;
	}



}
```

Appendix D2

The full WifiBlock arduino Code

```
/*
 * A simple sketch that uses WiServer to get the hourly weather data from LAX and prints
 * it via the Serial API
 */

#include <WiServer.h>
#include <string.h>
#include <Wire.h>   //Voor i2c communicatie

#define WIRELESS_MODE_INFRA 1
#define WIRELESS_MODE_ADHOC 2

#define senderLength 32

// Wireless configuration parameters ----------------------------------------
unsigned char local_ip[] = {192,168,0,3};	// IP address of WiShield
unsigned char gateway_ip[] = {192,168,0,1};	// router or gateway IP address
unsigned char subnet_mask[] = {255,255,255,0};// subnet mask for the local network
const prog_char ssid[] PROGMEM = {"Piloot en Paladina"};		// max 32 bytes

unsigned char security_type = 2;	// 0 - open; 1 - WEP; 2 - WPA; 3 - WPA2

// WPA/WPA2 passphrase
const prog_char security_passphrase[] PROGMEM = {"lotteislekker"};	// max 64 characters

// WEP 128-bit keys
// sample HEX keys
prog_uchar wep_keys[] PROGMEM = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, // Key 0
	0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, // Key 1
	0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
 0x00, // Key 2
                                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00  // Key 3
                                      };

// setup the wireless mode
// infrastructure - connect to AP
// adhoc - connect to another WiFi device
unsigned char wireless_mode = WIRELESS_MODE_INFRA;

unsigned char ssid_len;
unsigned char security_passphrase_len;
// End of wireless configuration parameters ----------------------------------------

//Parameter for GetRequest object ------\

// IP Address for of my local server
uint8 ip[] = {192,168,0,100}; //Mijn Mac Server
//uint8 ip[] = {127,0,0,1}; //MAMP host
boolean submitVal = false;
boolean receiveVal = false;

//Variables to set the getrequestValue.
char* getStartValue = "/library.php?v1=A&v2=348&v3=Re";
char* startValue= "/okay/variables.php?";
char* getVariableVal = "var1=34&var2=23";
char* getFullValue = "/library.php?v1=A&v2=348&v3=R2";//
char URL[32] = "/library.php?v1=A&v2=348&v3=Re"; // howmany van de receive event.

// A request that gets the latest METAR weather data for LAX
byte dataBack[senderLength] = "/library.php?v1=A&v2=348&v3=Re"; // De variable om data terug te sturen
byte dataBufferBack[senderLength];

GETrequest getVariables(ip, 8888, "localhost", getStartValue); // port voor normaal = 80 voorMAMP = 8888

// Time (in millis) when the data should be retrieved
long updateTime = 0;
int incomingByte = 0;
boolean Startup = false; //boolean to indicate the start of the wiShield
boolean wiShieldActive = false;  //Indicates if the wishield is active
byte isInactive[1] = {0};

void clearDataBuffer(void) {
   for(int i = 0; i < senderLength; i++) {
    dataBufferBack[i] = 0; //Putting everything of
   }

}
void clearSenderArray(void) {
     for(int i = 0; i < senderLength; i++) {
    dataBack[i] = 0; //Putting everything of

   }

}

// This is our page serving function that generates web pages
boolean sendMyPage(char* URL) {

 // Serial.println("Page printing begun");
  //Serial.println(URL);

  //  printStates();
  //  writeStates();
```

```
 //  changei2c(); // het veranderen van de waardes om door te kunen geven.

 //check whether we need to change the led state
 if (URL[1] == '?' && URL[2] == 'D' && URL[3] == 'I' && URL[4] == 'R') //url has a leading /
 {
   clearSenderArray();
  // Serial.println(URL[6]);
   switch(URL[6]) {
     case 'U':
           dataBack[0] = 255;
           break;
     case 'L':
           dataBack[1] = 255;
           break;
     case 'R':
           dataBack[2] = 255;
           break;
     case 'D':
           dataBack[3] = 255;
           break;
     case 'S':
           dataBack[4] =255;
           break;


   }
   dataBack[5] = 255;

   //after having change state, return the user to the index page.
  // WiServer.print("<HTML><HEAD><meta http-equiv='REFRESH' content='0;url=/'></HEAD></HTML>");
       //Serial.print("Nu Print ik wel");
    WiServer.print("<html><head><title>Led switch</title></head>");
    WiServer.print("<body>Controlling the robot <form method=GET><input type=submit name=DIR value=UP>");
    WiServer.print("</form><form method=GET><input type=submit name=DIR value=LEFT>");
    WiServer.print("<input type=submit name=DIR value=RIGHT></form>");
     WiServer.print("<form method=GET><input type=submit name=DIR value=DOWN></form>");
    WiServer.print("<form method=GET><input type=submit name=DIR value=STOP></form>");
     WiServer.print("</body>");
    WiServer.print("</html> ");
   return true;
 }

 if (strcmp(URL, "/") == false) //why is this not true? Het is dus waar als de string gelijk is
  {
    //Serial.print("Nu Print ik wel");
    WiServer.print("<html><head><title>Led switch</title></head>");
    WiServer.print("<body>Controlling the robot <form method=GET><input type=submit name=DIR value=UP>");
    WiServer.print("</form><form method=GET><input type=submit name=DIR value=LEFT>");
    WiServer.print("<input type=submit name=DIR value=RIGHT></form>");
     WiServer.print("<form method=GET><input type=submit name=DIR value=DOWN></form>");
    WiServer.print("<form method=GET><input type=submit name=DIR value=STOP></form>");
     WiServer.print("</body>");
    WiServer.print("</html> ");
     return true;
  }
  //return false;
}
// Function that prints data from the server
void printData(char* data, int len) {

 // Print the data returned by the server
 // Note that the data is not null-terminated, may be broken up into smaller packets, and
 // includes the HTTP header.
 int parsData = 0;
```

```
 int length = len;
 //Serial.print(length);
 for(int i = 1; i <len; i++) {
     if(data[i] == 'n' && data[i-1] == '<') {
        parsData = i-1;
        i = len;
     }
 }
//Vanaf de <nxt wordt er een code gegenereerd
 for(int i = parsData; i < (parsData + 28) && i < length; i++) {
   dataBack[i] = data[i];
    Serial.print(dataBack[i]);
 }
 receiveVal = true;
 //Serial.println(*dataBack);
}
//Function to give values back to NXT
void requestEvent()
{
   //printer();
   if(wiShieldActive) {
     Wire.send(dataBack, senderLength);
   } else {
     Wire.send(isInactive, 1);
   }
   //sender[5] = 0;
   //Serial.print("Request geweest");

}
// ---------------------------------------------------------------------------
// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void receiveEvent(int howMany)
{
 /* Serial.print("howMany: ");
  Serial.print(howMany);
  Serial.print(". "); */
  char c;
  int s;
  int availables = howMany-1;
 //The first char should be the register byte. I set this to 12 in the java code
  if(Wire.available()) {
    s = Wire.receive();
   // Serial.print(s);
  }
  /* Het protocol van de arduino en de NXT
  *  0 = wiShield init();
  *  1 = local_ip of wiShield;
  *  2 = gateway of wiShield;
  *  3 = subnet of wiShield;
  *  4 = ssid of wiShield;
              *  5 = security mode of the wishield. (0 = open, 1 = WPA, 2 = wpa2)
  *  6 = password of wireless network.
  *
  *  10 = check for new data out of the wiShield.
              *  11 = read data out of the wiShield
  *
              *  20 = set a new URL for getRequest wiShield. Linked to a php script that does something with it.
              *  21 = new IP adress  for the getRequest.
         */
  switch (s) {
    case 0:
       Startup = true;
       break;
```

```
case 1:
 while(0 < availables) // 12 is indicated in the send REgister .loop through all but the last Check if goed adress is
    {
      availables = Wire.available();
      char c = Wire.receive(); // receive byte as a character
      //Serial.print(availables, DEC);
      local_ip[(howMany-1)-availables] = c;
    }
   break;
case 2:
     while(0 < availables) // 12 is indicated in the send REgister .loop through all but the last Check if goed adress is
    {
      availables = Wire.available();
      char c = Wire.receive(); // receive byte as a character
      //Serial.print(availables, DEC);
      gateway_ip[(howMany-1)-availables] = c;
    }

    break;
case 3:
     while(0 < availables) // 12 is indicated in the send REgister .loop through all but the last Check if goed adress is
    {
      availables = Wire.available();
      char c = Wire.receive(); // receive byte as a character
      //Serial.print(availables, DEC);
      subnet_mask[(howMany-1)-availables] = c;
    }
   break;
case 4:
   /*  while(0 < availables) // 12 is indicated in the send REgister .loop through all but the last Check if goed adress is
    {
      availables = Wire.available();
      char c = Wire.receive(); // receive byte as a character
      //Serial.print(availables, DEC);
      ssid[(howMany-1)-availables] = c;
    }*/

    break;
case 5:
     while(0 < availables) // 12 is indicated in the send REgister .loop through all but the last Check if goed adress is
    {
      availables = Wire.available();
      char c = Wire.receive(); // receive byte as a character
      //Serial.print(availables, DEC);
      security_type = c;
    }
   break;
case 6:
  /* while(0 < availables) // 12 is indicated in the send REgister .loop through all but the last Check if goed adress is
    {
      availables = Wire.available();
      char c = Wire.receive(); // receive byte as a character
      //Serial.print(availables, DEC);
      security_passphrase[(howMany-1)-availables] = c;
    }*/

    break;
case 10:
   // Serial.print("Ik wordt geroepen");
   break;
case 11:

    break;
case 20:
```

```
      break;
   case 21:
       while(0 < availables) // 12 is indicated in the send REgister .loop through all but the last Check if goed adress is
      {
        availables = Wire.available();
        char c = Wire.receive(); // receive byte as a character
        //Serial.print(availables, DEC);
        ip[(howMany-1)-availables] = c;
      }
     break;

   case 12:
     while(0 < availables) // 12 is indicated in the send REgister .loop through all but the last Check if goed adress is
      {
        availables = Wire.available();
        char c = Wire.receive(); // receive byte as a character
        //Serial.print(availables, DEC);
        URL[(howMany-1)-availables] = c;
       // Serial.print(c);
        //Serial.print("-");     // print the character
      }
      getFullValue = URL;
     // Serial.println(getFullValue);
     submitVal = true;
    break;
  }
}

// A post Requests for the wiki. :D

void setup() {
   Serial.begin(57600);
   Wire.begin(27);                // join i2c bus with address 127
   Wire.onReceive(receiveEvent);
   Wire.onRequest(requestEvent);
}




void loop(){

  // send data only when you receive data:
  if (Serial.available() > 0) {

              // read the incoming byte:
              incomingByte = Serial.read();

              // say what you got:
            // Serial.println(getFullValue);
              Serial.print("I received: ");
              Serial.println(incomingByte, DEC);

     if(incomingByte == 65 && !getVariables.isActive()) {
          Serial.print("Doing a submit...");
         getVariables.setURL(getFullValue);
          getVariables.submit();
     }
     if(incomingByte == 66) {
       Startup = true;
     }
  }
  if(Startup && !wiShieldActive) {
      Serial.print("Active request");
```

```
  // Initialize WiServer (we'll pass NULL for the page serving function since we don't need to serve web pages)
  WiServer.init(sendMyPage);//sendMyPage);

  // Enable Serial output and ask WiServer to generate log messages (optional)

  WiServer.enableVerboseMode(true);
  // Have the processData function called when data is returned by the server
  getVariables.setReturnFunc(printData);
  wiShieldActive = true;
  Serial.print("Active gegaan");

 }
 if(wiShieldActive) {
    if(submitVal) {
       getVariables.setURL(getFullValue);
       getVariables.submit();
       submitVal = false;
    }
    if(receiveVal) {
      for(int i = 0; i < senderLength; i++) {
        Serial.print(dataBack[i]);
      }
      receiveVal = false;
    }
    // Run WiServer
    WiServer.server_task();
  }


 //delay(10);
}
```

Appendix E1

Lejos code: exploring the room.

```
import lejos.nxt.*;
import lejos.robotics.navigation.*;
//import java.lang.*;
public class exploreRoomv2 {

            static String URL = "/lib.php?v1=B&v2=0000000&v3=Re";
 static byte[] send = new byte[URL.length()];
            static TachoPilot pilot;
            static wiShield Arduino = new wiShield(SensorPort.S2);
 byte[] buf = new byte[32];

 public static void writeIntoArray(int distance) {
             //byte[] variable = new byte [4];
             int start = 21;
                        String sensVal = Integer.toString(distance);
                        start = 24 - sensVal.length();
             for(int i = 17; i<23; i++) {
                         send[i] = '0';
             }
                        for(int i = start ; i < start + sensVal.length(); i++){
                                   send[i] = (byte) sensVal.charAt(i-start);
```

```
		LCD.drawString(URL, 0, 2);
		}
public static void writeActionIntoArray(boolean action) {
		//byte[] variable = new byte [4];
				//String sensVal = Integer.toString(distance);
		if(action) {
				 send[28] = 'v';
				 send[29] = 'o';

		} else {
				 send[28] = 'd';
				 send[29] = 'r';
		}
				/*for(int i = 28 ; i < 28 + sensVal.length(); i++){
						send[i] = (byte) sensVal.charAt(i-21);
						//URL.charAt(i) = sensVal.charAt(i-21);
		}*/
		LCD.drawString(URL, 0, 2);
		}
public static void sendDataArduino() {
		Arduino.sendData(12, send, send.length);
		LCD.drawString("Bezig met Schrijven", 0, 7);
		try {
				 Thread.sleep(200);
		} catch (Exception e) {
		}
		LCD.drawString("Klaar met Schrijven", 0, 7);

		}
		public static void main(String[] args) {
				// TODO Auto-generated method stub
				pilot =  new TachoPilot((float) 41.4f,(float)120f, Motor.A, Motor.B);
				TouchSensor touch = new TouchSensor(SensorPort.S1);
				TouchSensor touch2 = new TouchSensor(SensorPort.S3);
		boolean drive = true;
		boolean turn = false;
		int turnAngle = 0;
		  // Arduino = new I2CSensor(SensorPort.S2);
		//Arduino.setAddress(27);
				Arduino.startWiShield();
		//byte[] buf = new byte[28];

		for(int i = 0; i <send.length; i++){
				 send[i] = (byte) URL.charAt(i);
		}
		pilot.reset();
				LCD.drawString("Is de Afstand", 0, 0);
		   LCD.drawInt((int) pilot.getTravelDistance(), 0, 1);
		   //writeIntoArray((int) pilot.getTravelDistance());

		try {
				 Thread.sleep(1200);
		} catch (Exception e) {
		}
		pilot.reset();
		   while(!Button.LEFT.isPressed()) {
				 if(drive) {

						pilot.forward();
		if (touch.isPressed() || touch2.isPressed()) {
			pilot.stop();
			writeActionIntoArray(drive);
			  writeIntoArray((int) pilot.getTravelDistance());
			  LCD.drawString("Is de Afstand", 0, 0);
```

```
                    LCD.drawString("      ",0,1); //Om te clearen
                      LCD.drawInt((int) pilot.getTravelDistance(), 0, 1);
                    sendDataArduino();
                       try {
                                Thread.sleep(4000);
                       } catch (Exception e) {
                       }
                    drive = false;
                    turn = true;
                    pilot.reset();
            }
          }
                        if(turn) {
                                 turnAngle = (int) (Math.random() * 180);
                                 pilot.rotate(turnAngle);
                                 writeActionIntoArray(drive);
                                 writeIntoArray(turnAngle);
                    LCD.drawString("De Hoek =", 0, 3);
                    LCD.drawString("       ", 0, 4);
                    LCD.drawInt(turnAngle, 0, 4);
                                 sendDataArduino();
                                 drive = true;
                                 try {
                                          Thread.sleep(4000);
                                 } catch (Exception e) {
                                 }
                                 try {
                                          Thread.sleep(1200);
                                 } catch (Exception e) {
                                 }
                                 turn = false;
                        }

    }
            }

}
```

Appendix E2

The WifiBlock code for the exploring the room example.

```
/*
 * A simple sketch that uses WiServer to get the hourly weather data from LAX and prints
 * it via the Serial API
 */

#include <WiServer.h>
#include <string.h>
#include <Wire.h>   //Voor i2c communicatie

#define WIRELESS_MODE_INFRA 1
#define WIRELESS_MODE_ADHOC 2

#define senderLength 32
```

```
// Wireless configuration parameters ----------------------------------------
unsigned char local_ip[] = {192,168,0,3};           // IP address of WiShield
unsigned char gateway_ip[] = {192,168,0,1};      // router or gateway IP address
unsigned char subnet_mask[] = {255,255,255,0}; // subnet mask for the local network
const prog_char ssid[] PROGMEM = {"Piloot en Paladina"};            // max 32 bytes

unsigned char security_type = 2;          // 0 - open; 1 - WEP; 2 - WPA; 3 - WPA2

// WPA/WPA2 passphrase
const prog_char security_passphrase[] PROGMEM = {"lotteislekker"};           // max 64 characters

// WEP 128-bit keys
// sample HEX keys
prog_uchar wep_keys[] PROGMEM = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c,
0x0d, // Key 0
                                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, // Key 1
                                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, // Key 2
                                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00  // Key 3
                                 };

// setup the wireless mode
// infrastructure - connect to AP
// adhoc - connect to another WiFi device
unsigned char wireless_mode = WIRELESS_MODE_INFRA;

unsigned char ssid_len;
unsigned char security_passphrase_len;
// End of wireless configuration parameters ----------------------------------------

//Parameter for GetRequest object ------\

// IP Address for of my local server
uint8 ip[] = {192,168,0,100}; //Mijn Mac Server
//uint8 ip[] = {127,0,0,1}; //MAMP host
boolean submitVal = false;
boolean receiveVal = false;

//Variables to set the getrequestValue.
char* getStartValue = "/library.php?v1=A&v2=348&v3=Re";
char* startValue= "/okay/variables.php?";
char* getVariableVal = "var1=34&var2=23";
char* getFullValue = "/library.php?v1=A&v2=348&v3=R2";//
char URL[32] = "/library.php?v1=A&v2=348&v3=Re"; // howmany van de receive event.
// A request that gets the latest METAR weather data for LAX
byte dataBack[senderLength] = "/library.php?v1=A&v2=348&v3=Re"; // De variable om data terug te sturen

GETrequest getVariables(ip, 8888, "localhost", getStartValue); // port voor normaal = 80 voorMAMP = 8888

void clearSenderArray(void) {
     for(int i = 0; i < senderLength; i++) {
    dataBack[i] = 0; //Putting everything of

   }

}

// This is our page serving function that generates web pages
boolean sendMyPage(char* URL) {

 // Serial.println("Page printing begun");
  //Serial.println(URL);
```

```
  //  printStates();
  //  writeStates();
  //  changei2c(); // het veranderen van de waardes om door te kunen geven.

  //check whether we need to change the led state
  if (URL[1] == '?' && URL[2] == 'D' && URL[3] == 'I' && URL[4] == 'R') //url has a leading /
  {
    clearSenderArray();
   // Serial.println(URL[6]);
    switch(URL[6]) {
      case 'U':
            dataBack[0] = 255;
            break;
      case 'L':
            dataBack[1] = 255;
            break;
      case 'R':
            dataBack[2] = 255;
            break;
      case 'D':
            dataBack[3] = 255;
            break;
      case 'S':
            dataBack[4] =255;
            break;


    }
    dataBack[5] = 255;

    //after having change state, return the user to the index page.
   // WiServer.print("<HTML><HEAD><meta http-equiv='REFRESH' content='0;url=/'></HEAD></HTML>");
        //Serial.print("Nu Print ik wel");
     WiServer.print("<html><head><title>Led switch</title></head>");
     WiServer.print("<body>Controlling the robot <form method=GET><input type=submit name=DIR value=UP>");
     WiServer.print("</form><form method=GET><input type=submit name=DIR value=LEFT>");
     WiServer.print("<input type=submit name=DIR value=RIGHT></form>");
      WiServer.print("<form method=GET><input type=submit name=DIR value=DOWN></form>");
     WiServer.print("<form method=GET><input type=submit name=DIR value=STOP></form>");
      WiServer.print("</body>");
     WiServer.print("</html> ");
    return true;
  }

  if (strcmp(URL, "/") == false) //why is this not true? Het is dus waar als de string gelijk is
   {
     //Serial.print("Nu Print ik wel");
     WiServer.print("<html><head><title>Led switch</title></head>");
     WiServer.print("<body>Controlling the robot <form method=GET><input type=submit name=DIR value=UP>");
     WiServer.print("</form><form method=GET><input type=submit name=DIR value=LEFT>");
     WiServer.print("<input type=submit name=DIR value=RIGHT></form>");
      WiServer.print("<form method=GET><input type=submit name=DIR value=DOWN></form>");
     WiServer.print("<form method=GET><input type=submit name=DIR value=STOP></form>");
      WiServer.print("</body>");
     WiServer.print("</html> ");
      return true;
   }
   //return false;
}
// Function that prints data from the server
void printData(char* data, int len) {

  // Print the data returned by the server
  // Note that the data is not null-terminated, may be broken up into smaller packets, and
```

```
  // includes the HTTP header.
  int parsData = 0;
  int length = len;
  //Serial.print(length);
  for(int i = 1; i <len; i++) {
      if(data[i] == 'n' && data[i-1] == '<') {
         parsData = i-1;
         i = len;
      }
  }
 //Vanaf de <nxt wordt er een code gegenereerd
  for(int i = parsData; i < (parsData + 28) && i < length; i++) {
    dataBack[i] = data[i];
     Serial.print(dataBack[i]);
  }
  receiveVal = true;
  //Serial.println(*dataBack);
}
//Function to give values back to NXT
void requestEvent()
{
    //printer();
    Wire.send(dataBack, senderLength);
    //sender[5] = 0;
    //Serial.print("Request geweest");

}
// ---------------------------------------------------------------------------
// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void receiveEvent(int howMany)
{
 /* Serial.print("howMany: ");
  Serial.print(howMany);
  Serial.print(". "); */
  char c;
  int s;
  int availables = howMany-1;
 //The first char should be the register byte. I set this to 12 in the java code
  if(Wire.available()) {
    s = Wire.receive();
  }
  if(s ==12) {
    while(0 < availables) // 12 is indicated in the send REgister .loop through all but the last Check if goed adress is
    {
      availables = Wire.available();
      char c = Wire.receive(); // receive byte as a character
      //Serial.print(availables, DEC);
      URL[(howMany-1)-availables] = c;
     // Serial.print(c);
      //Serial.print("-");     // print the character
    }
    getFullValue = URL;
   // Serial.println(getFullValue);
   submitVal = true;
  }
}

// A post Requests for the wiki. :D

void setup() {
    Serial.begin(57600);
    Wire.begin(27);                // join i2c bus with address 127
    Wire.onReceive(receiveEvent);
```

```
  Wire.onRequest(requestEvent);
 // Initialize WiServer (we'll pass NULL for the page serving function since we don't need to serve web pages)
 WiServer.init(sendMyPage);//sendMyPage);

 // Enable Serial output and ask WiServer to generate log messages (optional)

 WiServer.enableVerboseMode(true);
 // Have the processData function called when data is returned by the server
 getVariables.setReturnFunc(printData);
}


// Time (in millis) when the data should be retrieved
long updateTime = 0;
int incomingByte = 0;

void loop(){

 // send data only when you receive data:
 if (Serial.available() > 0) {

            // read the incoming byte:
            incomingByte = Serial.read();

            // say what you got:
           Serial.println(getFullValue);
            Serial.print("I received: ");
            Serial.println(incomingByte, DEC);


    if(incomingByte == 65 && !getVariables.isActive()) {
         Serial.print("Doing a submit...");
        getVariables.setURL(getFullValue);
         getVariables.submit();
    }
    if(incomingByte == 66) {
      getFullValue = "/library.php?v1=A&v2=210&v3=Vo";
    }
     if(incomingByte == 67) {
      getFullValue = "/library.php?v1=A&v2=10&v3=Li";
    }
 }
 if(submitVal) {
   getVariables.setURL(getFullValue);
   getVariables.submit();
   submitVal = false;
 }
 if(receiveVal) {
  for(int i = 0; i < senderLength; i++) {
    Serial.print(dataBack[i]);
  }
  receiveVal = false;
 }
 // Run WiServer
 WiServer.server_task();

 //delay(10);
}
```