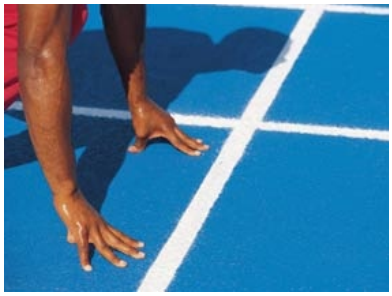
maXbox

# maXbox Starter 18



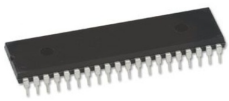
## Start with Arduino Programming V3.5

### 1.1 Physical Computing with RGB LED

Today we enter a topic in programming called embedded computing with the internet; we code a RGB LED light on a Arduino board with a breadboard on which we switch off or on the light by a browser on an android device with our own web server and their COM protocols too. Hope you did already work with the Starter 1 till 17 (especially the 17) at:

http://sourceforge.net/apps/mediawiki/maxbox/

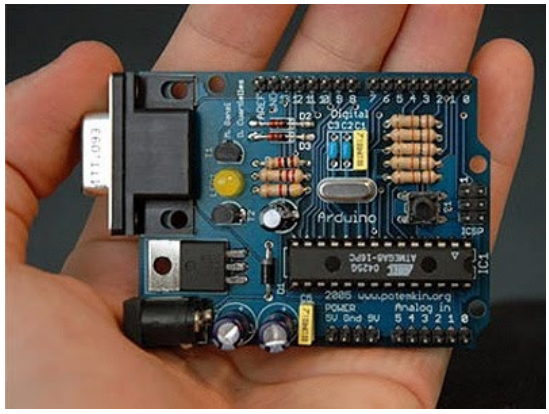
Arduino hardware is programmed using a Wiring-based language (syntax and libraries), similar to C++ and Object Pascal with some simplifications and modifications, and a Processing-based integrated development environment like Delphi or Lazarus with Free Pascal.
Current versions can be purchased pre-assembled; hardware design information is available for those who would like to assemble an Arduino by hand. Additionally, variations of the Italian-made Arduino—with varying levels of compatibility—have been released by third parties; some of them are programmed using the Arduino software or the sketch firmware.
The Arduino is what is known as a Physical or Embedded Computing platform, which means that it is an interactive system that through the use of hardware, firmware and software can interact with its environment.



This lesson will introduce you to Arduino and the Serial communication (see Tutorial 15). We will now dive into the world of serial communications and control our lamp from a browser to a web server by sending commands from the PC to the Arduino using a serial monitor with interface.

In our case we explain one example of a HTTP server which is an intermediate to the COM serial communication with the AVR based micro controller on Arduino[1].
Another Controller is the Delphi Controller. A Delphi Controller and the `DelphiDevBoard` were designed to help students, Pascal programmers and electronic engineers understand how to program micro controllers and embedded systems especially in programming these devices and targets (see links at the end).

This is achieved by providing hardware (either pre-assembled or as a DIY kit of components), using course material, templates, and a Pascal compatible cross-compiler and using of a standard IDE for development and debugging (Delphi, maXbox, Lazarus or Free Pascal).




Let's begin with HTTP (Hypertext Transfer Protocol) and TCP. TCP/IP stands for Transmission Control Protocol and Internet Protocol. TCP/IP can mean many things, but in most cases, it refers to the network protocol itself.
Each computer on a TCP/IP network has a unique address associated with it, the so called IP-Address. Some computers may have more than one address associated with them. An IP address is a 32-bit number and is usually represented in a dot notation, e.g. 192.168.0.1. Each section represents one byte of the 32-bit address. In maXbox a connection with HTTP represents an object.
In our case we will operate with the local host. It is common for computers to refer to themselves with the name local host and the IP number 127.0.0.1.

## 1.2 Get the Code

As you already know the tool is split up into the toolbar across the top, the editor or code part in the centre and the output window at the bottom. Change that in the menu `/view` at our own style.

☝ In maXbox you will start the web server as a script, so the web server IS the script that starts the Indy objects, configuration and a browser too (on board: `Options/Add_ons/Easy_Browser/.`

Before this starter code will work you will need to download maXbox from the website. It can be down-loaded from http://www.softwareschule.ch/maxbox.htm (you'll find the download maxbox3.zip on the top left of the page). Once the download has finished, unzip the file, making sure that you preserve the folder structure as it is.

[1] An Arduino board consists of an 8-bit Atmel AVR microcontroller, or an ARM cortex on the Due

If you double-click `maxbox3.exe` the box opens a default demo program. Test it with F9 / F2 or press **Compile** and you can open now the script example:

`443_webserver_arduino_rgb_light5.txt`

If you want to get the whole package including Arduino sketches too then try the zip-file:

http://sourceforge.net/projects/maxbox/files/Arduino/arduinopackage.zip/download

only sources:

http://www.softwareschule.ch/download/ardpac.zip

Now let's take a look at the code of this project. Our first line is

```
01 program Motion_HTTPServer_Arduino42_RGB_LED_Light;
```

☞ This example requires two objects from the classes: `TIdCustomHTTPServer` and `TComPort` so the second one is to connect and transport with the COM ports to Arduino (see below). `TComPort` by Dejan Crnila[2] are Delphi/C++ Builder serial communications components. It is generally easy to use for basic serial communications purposes, alternative to the TurboPower *ASYNCPro*.
It includes 5 components: `TComPort`, `TComDataPacket`, `TComComboBox`, `TComRadioGroup` and `TComLed`. With these tools you can build serial communication apps easier and faster than ever.
First we start with the web server and second we explain the COM port.
After creating the object we use first methods to configure our server calling Port and IP. The object makes a bind connection with the Active method by passing a web server configuration.

```
123 HTTPServer:= TIdCustomHTTPServer.Create(self);
```

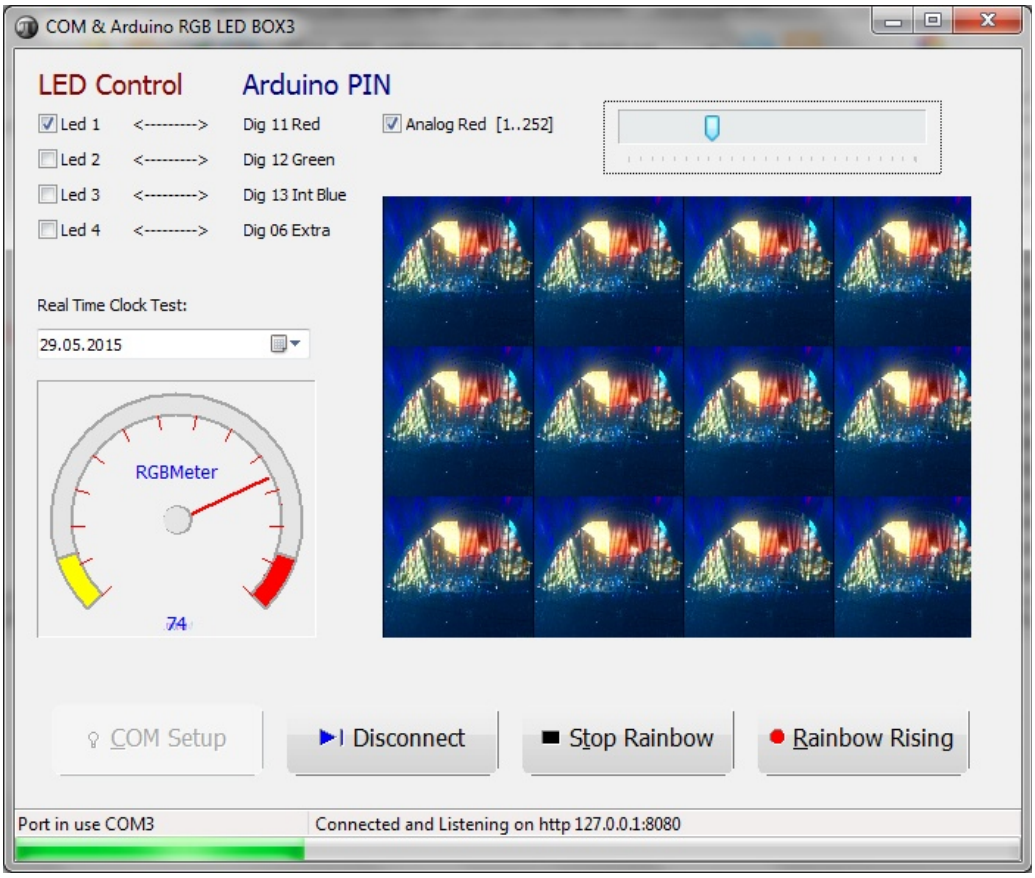
So the object `HTTPServer` has some methods and properties like `Active` you can find in the `TIdCustomHTTPServer.pas` unit or `IdHTTPServer` library. A library is a collection of code or classes, which you can include in your program. Once a unit is tested it's stable to use.

But where does this function come from?

They use the windows 32-bit application programming interface, which basically means interacting with Win operating systems and libraries such as Win XP or 7 or 8 and the concerning DLL.
So most of the calls are valuable from a DLL:

```
function  MyNETConnect(hw: hwnd; dw: dword): Longint;
    external 'WNetConnectionDialog@Mpr.dll stdcall';
```

[2] http://sourceforge.net/projects/comport/




1: The GUI of the Win App

Indy is designed to provide a very high level of abstraction. Much more stuff or intricacies and details of the TCP/IP stack are hidden from the Indy programmer. A typical Indy client session looks like this:

```
with IndyClient do begin
  Host:= 'zip.pbe3.com'; // Host to call
  Port:= 6000; // Port to call the server on
  Connect; // get something to do with it
end;
```

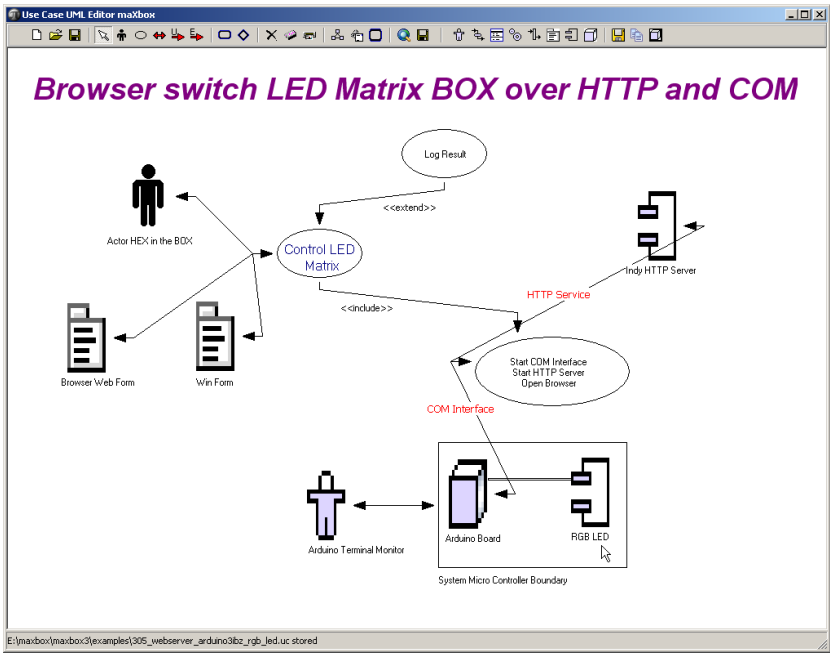
Indy is different than other so called winsock components you may be familiar with. If you've worked with other components, the best approach for you may be to forget how they work. Nearly all other components use non-blocking (asynchronous) calls and act asynchronously. They require you to respond to events, set up state machines, and often perform wait loops.

☝In facts there are 2 programming models used in TCP/IP applications. Non Blocking means that the application will not be blocked when the application socket read/write data. This is efficient, because your application don't have to wait for a connection. Unfortunately, it is complicated to develop.

Like a web socket. According to the draft specification, the Web Socket protocol enables two-way communication between a user agent running untrusted code running in a controlled environment layered over TCP to a remote host that has opted-in to communications from that code.

The goal of this technology is to provide a mechanism for browser-based applications that need two-way communication with servers that does not rely on opening multiple HTTP connections (e.g. using XMLHttpRequest or `iframes` and long polling).

Translated into human language, using Web Sockets, the (web)server can now initiate communication and send data to the client (browser) without being asked in a HTTP. This happens after a trusty channel of communication is established over a TCP connection.




2: The Use Case

So let's get back to our HTTP Create in line 123. In line 131 and 132 you see port and IP address configuration of a `Const`, instead of IP you can also set a host name as a parameter.

```
126   with HTTPServer do begin
127     if Active then Free;
128     if not Active then begin
129       Bindings.Clear;
130       bindings.Add;
131       bindings.items[0].Port:= APORT;
132       bindings.items[0].IP:= IPADDR; //'127.0.0.1'; 192.168.1.53'
133       Active:= true;
134       onCommandGet:= @HTTPServerGet;
135       PrintF('Listening HTTP on %s:%d.',[Bindings[0].IP,Bindings[0].Port]);
136     end;
```

☞ **Host names** are "human-readable" names for IP addresses. An example host name is max.kleiner.com, the www is just a convention and not necessary. Every host name has an equivalent IP address, e.g. www.hower.org = 207.65.96.71.


3: A few gadgets for Arduino

The full target of the request message is given by the URL property. Usually, this is a URL that can be broken down into the protocol (HTTP), Host (server system), script name (server application), path info (location on the host), and a query.

So far we have learned little about HTTP and host names. Now it's time to run our program at first with F9 (if you haven't done yet) and learn something about GET and HTML. The program (server) generates a standard HTML output or other formats (depending on the MIME type) after downloading with GET or HEAD.

So our command to shine on a LED is ../LED and to switch off is ../DEL (`127.0.0.1:8000/LED`).
Those are GET commands send with the browser, or /R for Red or /G for Green.
The first line identifies the request as a `GET`. A `GET` request message asks the Web server application to return the content associated with the URI that follows the word `GET`.
The following shows the magic behind in the method `HTTPServerGet()`:

```
43 procedure HTTPServerGet(aThr: TIdPeerThread; reqInf: TIdHTTPRequestInfo;
44                                               respInf: TIdHTTPResponseInfo);
```

One word concerning the thread: In the internal architecture there are 2 threads categories.
First is a listener thread that "listens" and waits for a connection. So we don't have to worry about threads, the built in thread `TIdPeerThread` will be served by Indy through a parameter:

```
54 if uppercase(localcom) = uppercase('/LED') then begin
55    cPort.WriteStr('1')
56    writeln(localcom+ ': LED on');
57    RespInfo.ContentText:= getHTMLContentString('LED is:  ON');
58 end else
59 if uppercase(localcom) = uppercase('/DEL') then begin
60   cPort.WriteStr('A');
61   writeln(localcom+ ': LED off');
62   RespInfo.ContentText:= getHTMLContentString('LED is:  OFF')
63 end;
```

HTTP request messages contain many headers that describe information about the client, the target of the request, the way the request should be handled, and any content sent with the request. Each header is identified by a name, such as "Host" followed by a string value. It then does a request.

☞You can also switch with F5 in a browser to switch LED on and off:

```
69    webswitch:= NOT webswitch;
70      if webswitch then begin
71        cPort.WriteStr('1') //goes to Arduino
72        RespInfo.ContentText:= getHTMLContentString('LED is:  ON Switch');
73      end else begin
74        cPort.WriteStr('A');
75        RespInfo.ContentText:= getHTMLContentString('LED is:  OFF Switch')
76      end
77     end
```

One of a practical way to learn much more about actually writing HTML is to get in maXbox editor and load or open a web-file with the extension html. Or you copy the output and paste it in a new maXbox instance. Then you click on the context menu and change to `HTML Syntax`!
In this mode the PC is a master and executes the control algorithm while the Arduino or Delphi Controller acts as an interface slave and follows commands coming from the PC or browser through its RS232 port. Each RGB field in these records reflects a state of the sensors and actuators of the LED in those sense only actors as LED light are in use.
On menu `/View/VTerminal` you do have a terminal monitor to test your script.
A running Arduino (M485A) monitor server will accept commands on input through a RS232 port:

```
  if (val=='1'){
      digitalWrite(ledPin11,HIGH); }
    else if (val=='A'){
      digitalWrite(ledPin11,LOW);
     }
  if (val=='2'){
      digitalWrite(ledPin12,HIGH); }
    else if (val=='B'){
      digitalWrite(ledPin12,LOW);
     }
```


4: The Browser controls

When the browser starts from script the server is ready for commands to pass chars to the serial communication. When a the server application finishes with our client request, it lights the LED and constructs a page of HTML code or other MIME content, and passes the result back (via the server in `TIdHTTPResponseInfo` ) to the client for display.

```
61    writeln(localcom+ ': LED on');
62    RespInfo.ContentText:= getHTMLContentString('LED is:  ON');
```

Have you tried the program, it's also possible to test the server without Arduino or a browser. When you run this code from the script `102_pas_http_download.txt` you will see a content (first 10 lines) of the site in HTML format with the help of the method `memo2.lines.add`:

```
begin
  idHTTP:= TIdHTTP.Create(NIL)
  try
    memo2.lines.text:= idHTTP.Get2('http://127.0.0.1')
    for i:= 1 to 10 do
      memo2.lines.add(IntToStr(i)+' :'+memo2.lines[i])
  finally
    idHTTP.Free
end
```

☝The Object `TIdHTTP` is a dynamically allocated block of memory whose structure is determined by its class type. With the method `Get1` you can download files.

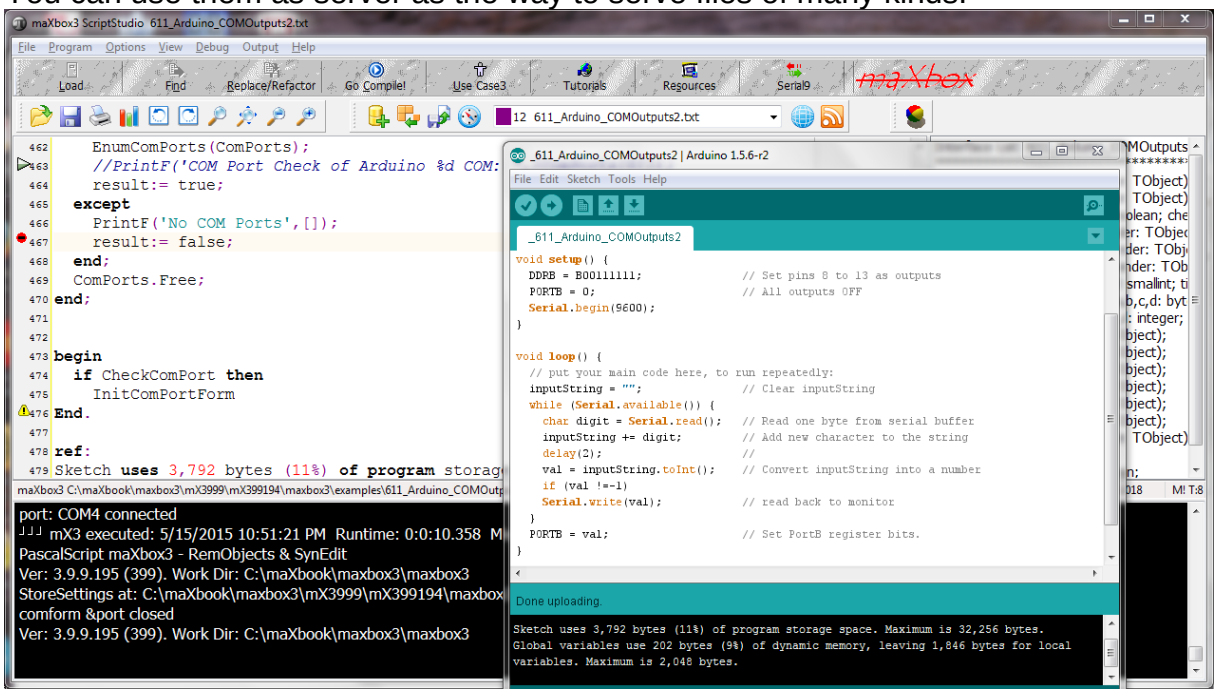
```
11 begin
12  myURL:= 'http://www.softwareschule.ch/download/maxbox_examples.zip';
13  zipStream:= TFileStream.Create('myexamples2.zip', fmCreate)
14  idHTTP:= TIdHTTP.Create(NIL)
15  try
16    idHTTP.Get1(myURL, zipStream)
```

Of course a lot of lines to get a file from the web - try it shorter with the short magic function `wGet()`:

```
wGet('http://www.softwareschule.ch/download/maxbox_starter17.pdf','mytestpdf.pdf');
```

It downloads the entire file into memory if the data is compressed (Indy 9 does not support streaming decompression for HTTP yet). Next we come closer to the main event of our web server in line 40, it's the event `onCommandGet` with the corresponding event handler method `@HTTPServerGet()` and one object of `TidPeerThread`.

You can use them as server as the way to serve files of many kinds!




5: maXuino Together Settings

## 1.3 Serial Line

Please read more about serial coding in Tutor 15! The serial communications line is simply a way for the Arduino to communicate with the outside world, in this case to and from the PC (via USB) and the Arduino IDE's Serial Monitor or from the uploaded code to I/O Board back.
Virtual COM port (VCP) drivers cause the USB device to appear as an additional COM port available to the PC. Application software can access the USB device in the same way as it would access a standard COM port.
We just create and configure our COM settings (depends in which COM port a USB hub works).

```
procedure TForm1_FormCreateCom(Sender: TObject);
begin
  cPort:= TComPort.Create(self);
  with cPort do begin
    BaudRate:= br9600;
    Port:= COMPORT; //'COM3';
    Parity.Bits:= prNone;
    StopBits:= sbOneStopBit;
    DataBits:= dbEight;
  end;
```

The Arduino can be used to develop stand-alone interactive objects or it can be connected to a computer to retrieve or send data to the Arduino and then act on that data (e.g. send sensor data out to the web or write data on a control LED).
Now we change to the Arduino editor to explain how he handles our commands (chars).
`Serial.begin` tells Arduino to start serial and the number within the parenthesis, in this case 9600, sets the baud rate (chars per second) that the serial line will communicate at.

```
int val = 0;          // variable to store data from the serial port
int ledPin11 = 11;    // LED connected to digital pin 11 or the inbuilt 13!

void setup() {
  pinMode(ledPin11,OUTPUT);   // declare a LED's pin as output mode
  erial.begin(9600);          // connect to serial port
  ..}
```

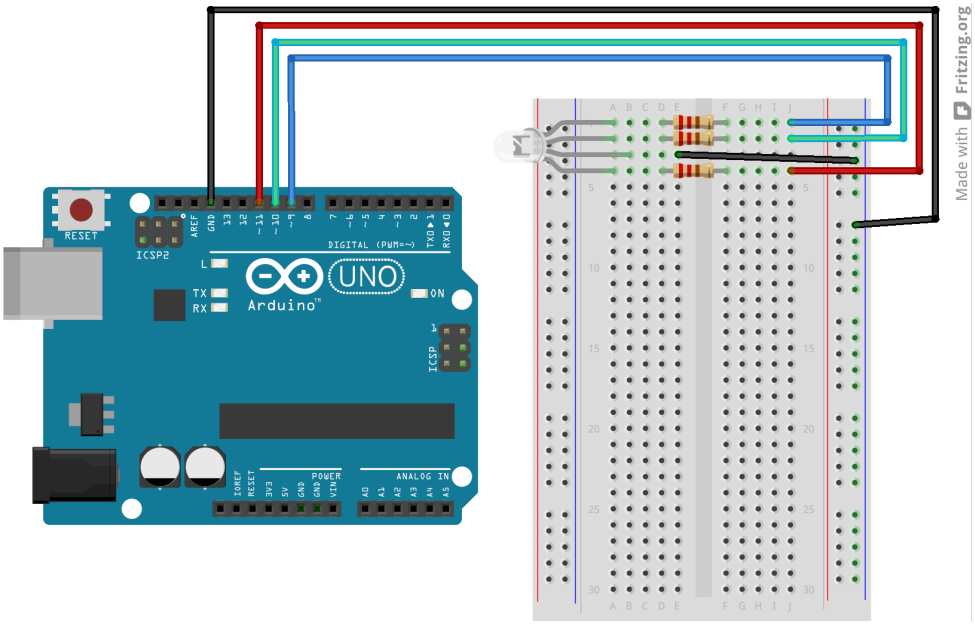
In the main loop we have an “if statement”. The condition it is checking the value in (`Serial.read`). The `Serial.available` command checks to see if any characters have been sent down the serial line. If any characters have been received then the condition is met and the code within the “if statements” code block is now executed, you see if ‘1’ then ON and if ‘A’ then OFF.
The condition of checking is simply a char it’s up to you to code a protocol of your own☺.

```
void loop () {
  val = Serial.read();      // read on the serial port
  if (val !=-1){
    if (val=='1'){
      digitalWrite(ledPin1,HIGH);
    }
    else if (val=='A'){
      digitalWrite(ledPin1,LOW);
      }
```

`Serial.print("Data entered: ");` and this is our way of sending data back from the Arduino to the PC. In this case the print command sends whatever is within the parenthesis to the PC, via the USB cable, where we can read it in the monitor window or in maXbox.

## 1.4 Bread Board Flow

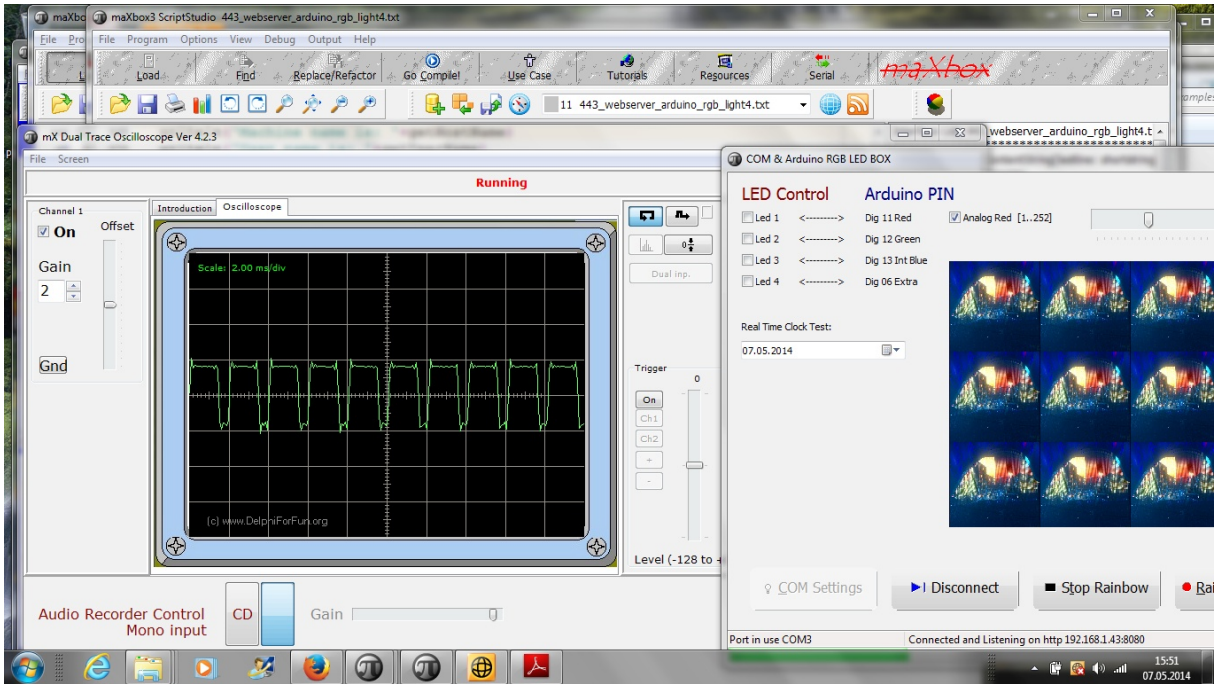
At last but not least some words about breadboards and electronics. A breadboard (or protoboard) is usually a construction base for prototyping devices of electronics. The term "breadboard" is commonly used to refer to a solder less breadboard (plug board).
Next time you may print out the breadboard on a 3D printer ;-).
With the breadboard you prepared above or below, add three wires for power to RGB light and one for ground GND for your AVR controller.
Place 3 resistors and LED as shown. Make sure the longest leg of the LED is to GND, connected to the minus. The resistors don’t have a direction, so it doesn’t matter which way it goes in.




6: Breadboard Settings

If you're using a standard breadboard, you'll need to use wires to reach the Arduino. Run 3 wires (red, green and blue) to the pin sockets on the Arduino. Run the other wire (black) to one of the GND sockets on the Arduino. The colours aren't essential but they will help you remember what the wires are connected to and black is a convention for ground GND!

Once the connection to a client socket is completed, the server connection is indistinguishable from a client connection. Both end points have the same capabilities and receive the same types of events. Only the listening connector is fundamentally different, as it has only a single endpoint.




7: PWM Signal measure in Oscilloscope

Sockets provide an interface between your network server or client application and a

networking software. You must provide an interface between your application and clients that use it.

Sockets let your network application communicate with other systems over the network. Each socket can be viewed as an endpoint in a network connection. It has an address that specifies:

- The system on which it is running.
- The types of interfaces it understands.
- The port it is using for the connection.

A full description of a socket connection includes the addresses of the sockets on both ends of the connection. You can describe the address of each socket endpoint by supplying both the IP address or host and the port number.

In the next line we just start a browser to test our server in a so called frame work flow ☺

```
34 procedure letOpenBrowser;
35 // TS_ShellExecuteCmd = (seCmdOpen,seCmdPrint,seCmdExplore);
36 begin
37 //ShellAPI.ShellExecute(Handle,PChar('open'),'http://127.0.0.1:80/',Nil,Nil,0);
38   S_ShellExecute('http:'+IPADDR+':'+IntToStr(APORT)+'/','',seCmdOpen)
39 end;
```

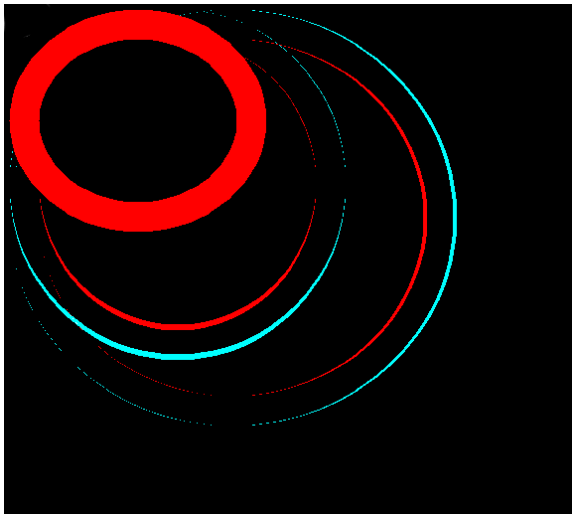
 Try to change the IP address in line 132 of `IP:= IPADDR` with a DHCP or dynDNS address, so you can reach Arduino from an Android, but change also settings.

Some notes at last about firewalls or proxy-servers. It depends on your network infrastructure to get a file or not, maybe you can't download content cause of security reasons and it stops with Socket-Error # 10060 and a time out error.
Furthermore, it also depends on the firewall in use at both ends. If it's automatic and recognises data that needs a response automatically it will work. It needs an administrator to open ports etc. you're stuffed or configured.

Hope you did learn in this tutorial the theme of Arduino with a web server.
The Arduino is an amazing device and will enable you to make anything from interactive works of art to robots. With a little enthusiasm to learn how to program the

Arduino and make it interact with other components a well as a bit of imagination, you can build anything you want.
The Arduino can also be extended with the use of Shields which circuit boards are containing other devices (e.g. GPS receivers, LED Cubes, LCD Displays, Sneakers, MIDI Synthesizers, Ethernet connections, etc.) that you can simply slot into the top of your Arduino to get extra functionality.

The Arduino board is made of an Atmel AVR microprocessor, a crystal or oscillator (basically a crude clock that sends time pulses to the micro-controller to enable it to operate at the correct what type of Arduino you have, you may also have a USB connector to enable it to be connected to a PC or Linux to upload or retrieve data. The board exposes the micro-controllers I/O (Input/Output) pins to enable you to connect those pins to other circuits, buses or to sensors, etc.



Feedback @
max@kleiner.com

Literature: Kleiner et al., Patterns konkret, 2003, Software & Support
Links of maXbox, Web of Things, Arduino and Indy:

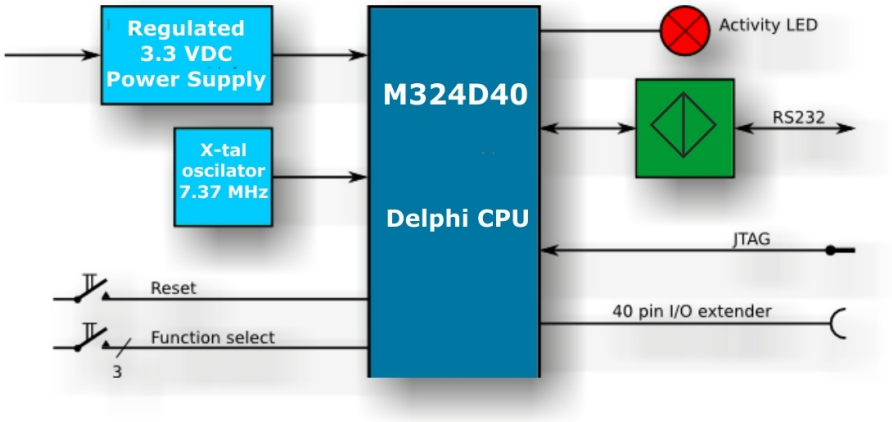
http://www.softwareschule.ch/download/webofthings2013.pdf

http://www.softwareschule.ch/download/codesign_2015.pdf

http://www.softwareschule.ch/maxbox.htm
http://www.indyproject.org/Sockets/index.EN.aspx
http://en.wikipedia.org/wiki/Arduino
http://fritzing.org/

http://sourceforge.net/projects/maxbox
http://sourceforge.net/projects/delphiwebstart
http://www.blaisepascal.eu/index.php?actie=./subscribers/UK_Electronics_Department
http://www.blaisepascal.eu/subscribers/vogelaar_elctronics_info_english.php

## 1.5 Appendix Delphi Controller




http://www.blaisepascal.eu/index.php?actie=./subscribers/UK_Electronics_Department

## 1.6 Delphi Schema