

maXbox

# maXbox Starter 30



## Start with Web of Things V2

### 1.1 Physical Computing with a lot of Things

Today we jump into a complete new topic in programming called web of things or embedded computing with the internet;

There are three main topics in here. First technologies – simply put, this part is mainly for early adopters. It's about coding, developing toys, plugging in kettles on the web (and we and many others actually did that!).

The second part is about new ideas, prototyping and new technologies that are in the lab. It's about research papers, and software philosophy, and about researchers worldwide. Third part is about end-users and products. That's the heaviest part of all.

I will show the topic with 6 examples:

- RGB LED in Arduino with an Oscilloscope
- Generating QR Code
- 3D Printing
- Web Video Cam
- Digi Clock with Real Time Clock
- Android SeekBar to Arduino LED Matrix

It's about products that exist, or at least should exist (or should not!). It's about marketing, fun (functionality) and new ideas out there.

As more "things" on planet Earth are converted to the inventory set of digitally connected Internet devices, the roles and responsibilities of web developers and technology managers will need to evolve.

So we code a RGB LED light on the Arduino board and 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 29 (especially the 18) 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 slight 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 in AVR. 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-shields and software can interact with its environment.

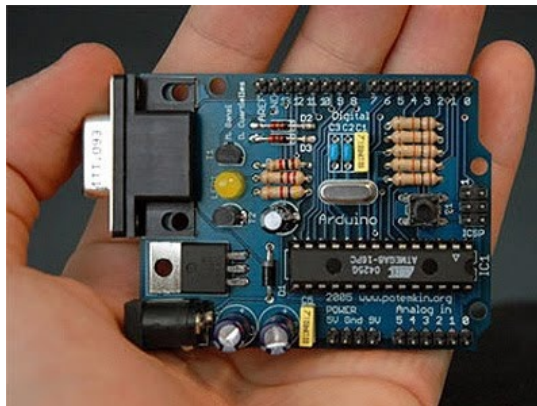


This lesson will introduce you to Arduino and the Serial communication (see Tutorial 15). We will now dive into the world of serial communications and control our lamp from a browser to a web server by sending commands from the PC to the Arduino using the Serial Monitor and Interface.

In our case we explain one example of a HTTP server which is an intermediate to the COM serial communication with the AVR micro controller on Arduino<sup>1</sup>.

Another Controller is the Delphi Controller. The Delphi Controller and 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 Appendix).

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 Indy 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.

👉 When HTTP is used on the Internet, browsers like Firefox on Android act as clients and the application that is hosting the website like softwareschule.ch acts as the server.

---

<sup>1</sup> An Arduino board consists of an 8-bit Atmel AVR [microcontroller](#), or an ARM cortex on the Due

### 1.1.1 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. If you double-click `maXbox3.exe` the box opens a default demo program. Test it with F9 / F2 or press **Compile** and you should hear a sound. So far so good now we'll open the examples:

```
443_webserver_arduino_rgb_light4.txt
102_pas_http_download.txt //if you don't use a browser
```

If you can't find the two files try also the zip-file loaded from:

[http://www.softwareschule.ch/download/maxbox\\_internet.zip](http://www.softwareschule.ch/download/maxbox_internet.zip) or direct as a file  
[http://www.softwareschule.ch/examples/443\\_webserver\\_arduino\\_rgb\\_light4.txt](http://www.softwareschule.ch/examples/443_webserver_arduino_rgb_light4.txt)

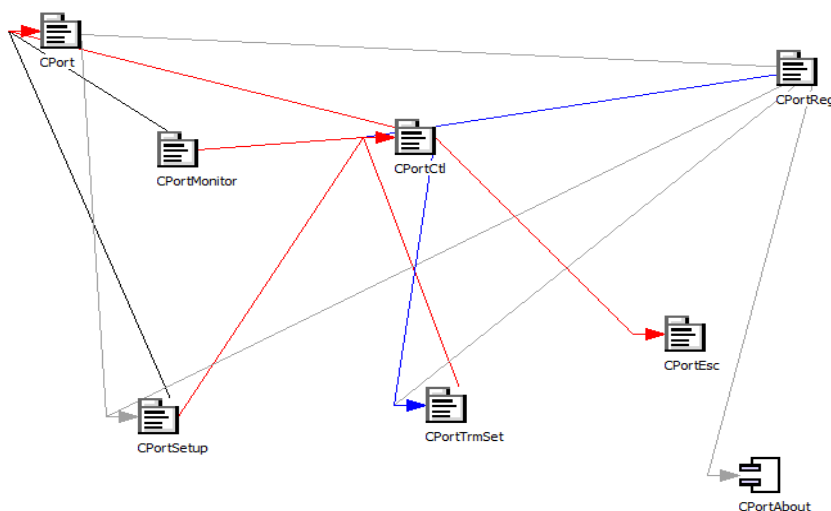
Now let's take a look at the code of this project. Our first line is

```
01 program Motion_HTTPServer_Arduino42_RGB_LED_Light;
```

We have to name the play, means the program's name is above.

👉 This example requires two objects from the classes: `TIdCustomHTTPServer` and `TComPort` so the second one is a package to connect and transport with the COM Ports to Arduino (see UML below).

`TComPort` by Dejan Crnila<sup>2</sup> are Delphi/C++ Builder serial communications components. It is generally easy to use for basic serial communications purposes, alternative to the TurboPower `ASYNCPPro` or `SynaSer` of Synapse.



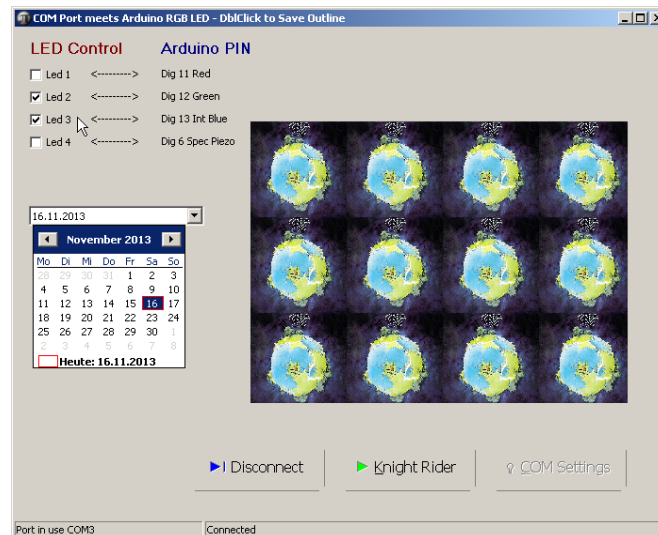
It includes 5 components: `TComPort`, `TComDataPacket`, `TComComboBox`, `TComRadioGroup` and `TComLed`. With these tools you can build serial communication apps easier and faster than ever. First we start with the web server and second we explain the COM port.

<sup>2</sup> <http://sourceforge.net/projects/comport/>

After creating the object in line 122 we use the 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.

```
122 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. By storing your commonly used code in a library, you can reuse code; Once a unit is tested it's stable to use.

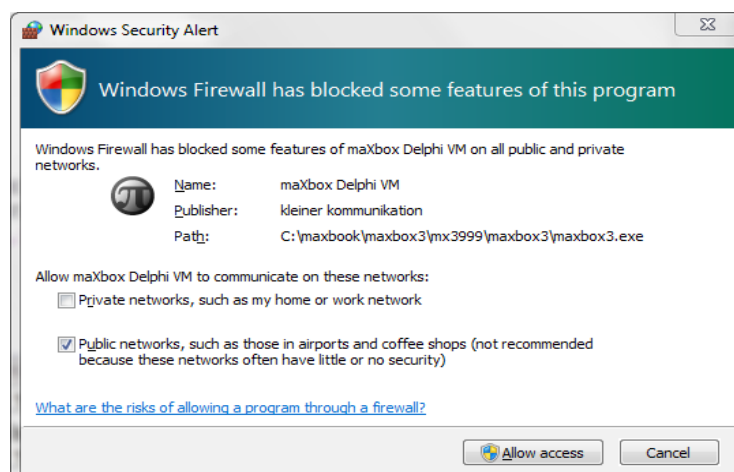


1: The GUI of the Win App

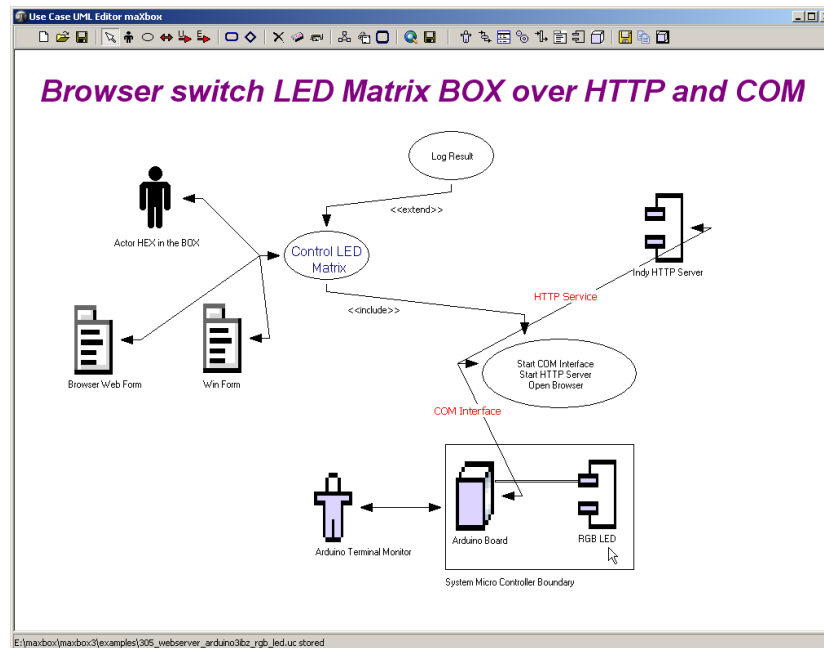
Indy is designed to provide a very high level of abstraction. Much more stuff or intricacies and details of the TCP/IP stack are hidden from the Indy programmer. A typical Indy client session looks like this:

```
with IndyClient do begin
    Host:= 'zip.pbe.com'; // Host to call
    Port:= 6000; // Port to call the server on
    Connect; // get something to do
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 test Indy. 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 app socket read/write data. This is efficient, because your app don't have to wait for connections. Unfortunately, it is complicated.

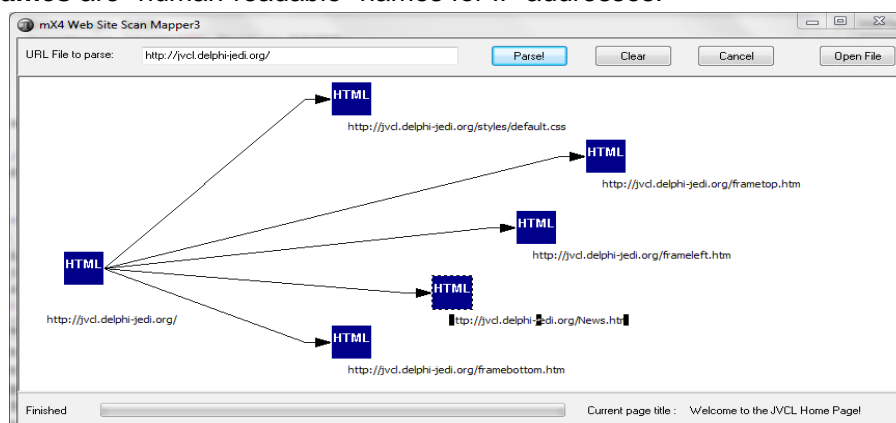


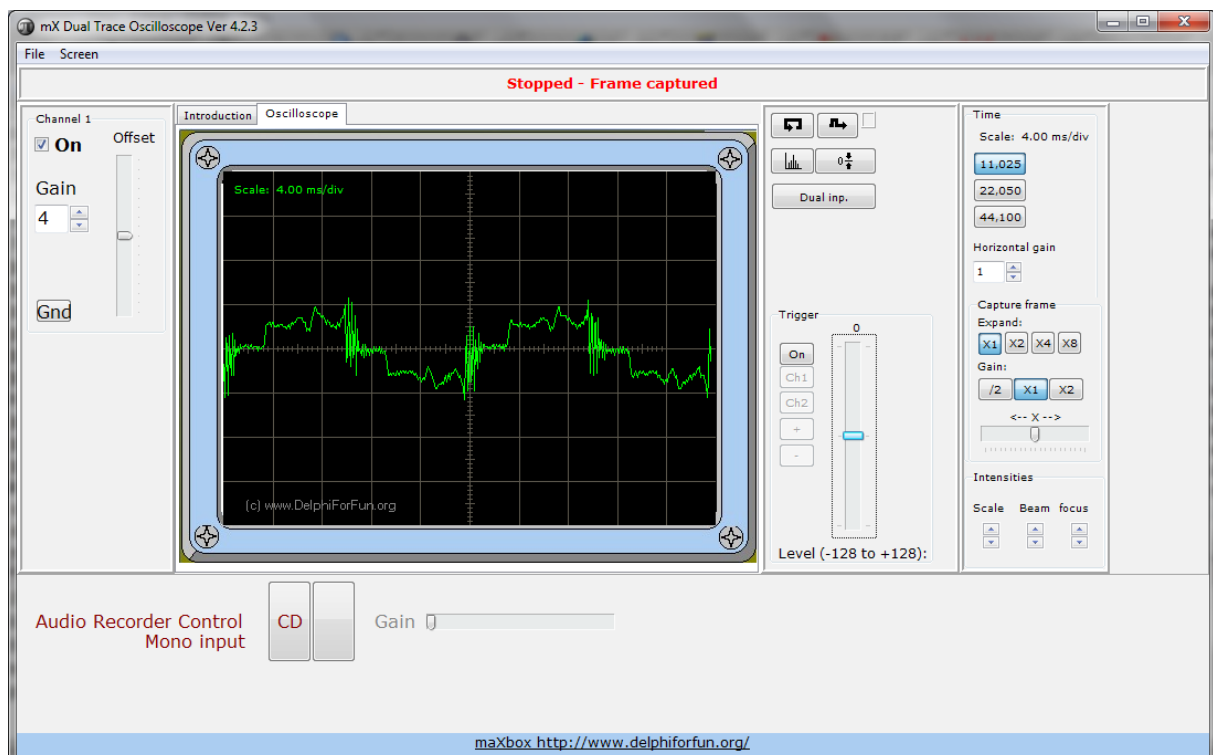
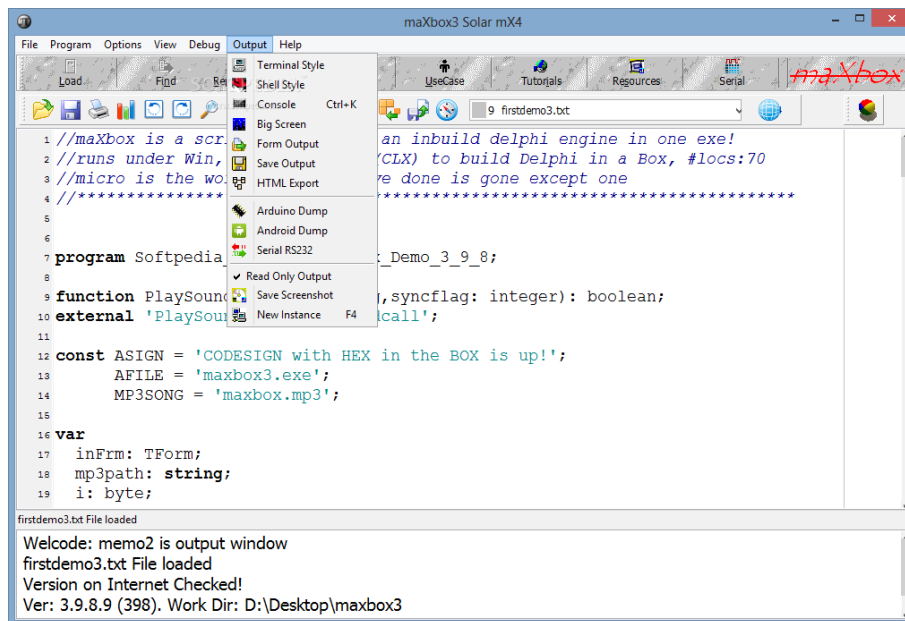
2: The Use Case of the App

Let's get back to our Create in line 125. In line 131 and 132 you see port and IP address config of a const in line 14, instead of IP you can set 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:= GetHostIP; //IPADDR; '127.0.0.1' or 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.





With this oscilloscope you can measure traffic or micro controllers in /Options/..

Host names are used both to make it easier on us humans, and to allow a computer to change its IP address without causing all of its potential clients (callers) to lose track of it.

Often called "URL or links" because a host address is normally inserted at the top of the browser as one of the first items to look at for searching the web.

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 your 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 `F5` to switch (`127.0.0.1:8080/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.WriteString('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.WriteString('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. When an HTML hypertext link is selected (or the user otherwise specifies a URL), the browser collects information about the protocol, the specified domain, the path to the information, the date and time, the operating environment, the browser itself, and other content information. It then does a request.

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

```
69     webswitch:= NOT webswitch;
70     if webswitch then begin
71         cPort.WriteString('1') //goes to Arduino for Red
72         RespInfo.ContentText:= getHTMLContentString('LED is:  ON Switch');
73     end else begin
74         cPort.WriteString('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 the `maXbox` editor and load or open a web-file with 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 the master and executes the control code while the Arduino or Delphi Controller acts as an interface slave and follows the commands coming from the PC or Browser through its RS232 port. Each RGB field in these records reflects the state of the sensors and actuators of the RGB LED in those sense only actors as LED light are in use. The running Arduino or M485A monitor server will accept read and write commands on the input through the RS232 port as follows:



```

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

```



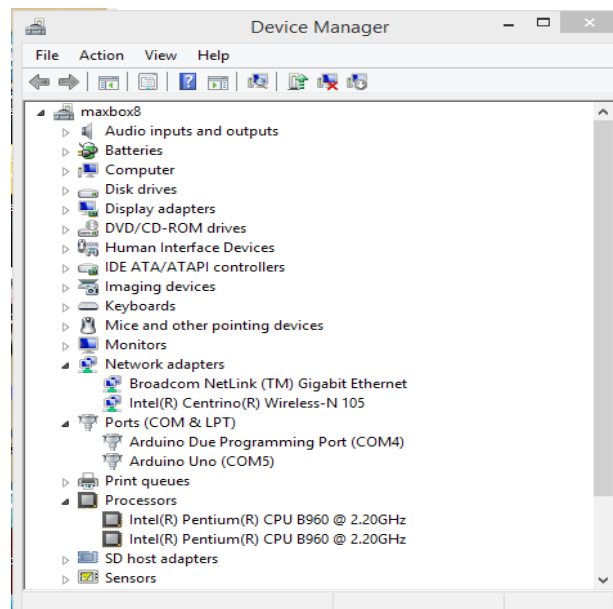
### 3: A Browser set

When the browser starts from script the server is ready for commands to pass as chars in line 59 or 60 to 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 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. The **Compile** button is also used to check that your code is correct, by verifying the syntax before the program starts. Another way to check syntax before run is F2 or the **Syntax Check** in menu Program. 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 a help of 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. Each object has a unique copy of every field defined in the class, but all instances of a class share the same methods. 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)

```

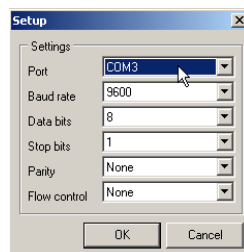
Of course a lot of lines to get a file from the web try it shorter with the magic function `wGet()`:

```

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

```

It downloads the entire file into memory if the data is compressed (Indy does not support streaming decompression for HTTP yet). Next we come closer to the main event of our web server, it's the event `onCommandGet` with the corresponding event handler method `@HTTPServerGet()` and one object of `TIdPeerThread`. You can use them as server to serve files of many kinds!



4: COM Port Settings

### 1.1.2 Serial Line

Please read more about serial coding in tutorial 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. We just create and configure our COM Settings (depends on which COM port the USB hub works) or try it with the app in picture 1 on the button "COM Settings":

```

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

```

```
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 the code site to the Arduino editor to explain how he handle our commands (ASCII chars).

`Serial.begin` tells the Arduino to start serial communications and the number within the parenthesis, in this case 9600, sets the baud rate (characters 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 inbuilt 13!

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

In the main loop we have an “if statement”. The condition 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 of the serial port
  if (val !=-1){
    if (val=='1'){
      digitalWrite(ledPin11,HIGH);
    }
    else if (val=='A'){
      digitalWrite(ledPin11,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 a serial monitor window or in maXbox (line 223).

But what about with analogue data to dime our LED; not just switch off and on?

The analog functions are a little different from the digital ones. First they do not need to be set up using `pinMode()` beforehand. Second, we are dealing with integer numbers that range from 0 to 255 or 0 to 1023, rather than digital states like HIGH and LOW!

```
procedure TF_trackSpeedChange(Sender: TObject);
var aout: string;
    aout2: char;
begin
  if chk_led5.checked then begin
    writeln(intToStr(tb.Position)); //for debug
    aout2:= Chr(tb.Position);      //!< ASCII byte by byte 0..255 with Chr()
    cPort.WriteString(aout2);
    cPort.ReadStr(aout,1);
    writeln('Analog back from ardu: '+aout);
```

The function `analogWrite()` on the opposite will allow us to access the pulse width modulation hardware on the Arduino micro controller. The basic syntax for this goes like this:

```
val = Serial.read();           // read serial port
    analogWrite(ledPin11, val); // val is of int, a cast of ASCII code!
    }
analogWrite(pin, duty cycle)
```

In using this function I need to give it two parts of information. The first is the pin number, which can only include one of the following pins on the Arduino Uno: 3, 5, 6, 9, 10, and **11**. These are marked as PWM (pulse width modulation) on the interface board and I have measuring that cycle with the oscilloscope at 500 Hz, so you can do:

[http://www.softwareschule.ch/images/maXbox\\_arduino\\_osci.png](http://www.softwareschule.ch/images/maXbox_arduino_osci.png)

But what is PWM? It approximates an analog signal by turning on and off a pin very quickly in the so called process pulse width modulation (PWM).

☞ This happens so fast it is nearly imperceptible to the eye, instead looking like a dim LED or a slow motor in our script! By turning on and off the pin very quickly it is possible to simulate a dim LED, just as we did with PWM.

In Line 343 we find another function of the runtime library of Indy:

```
343 Writeln(DateTimeToInternetStr(Now, true))
```

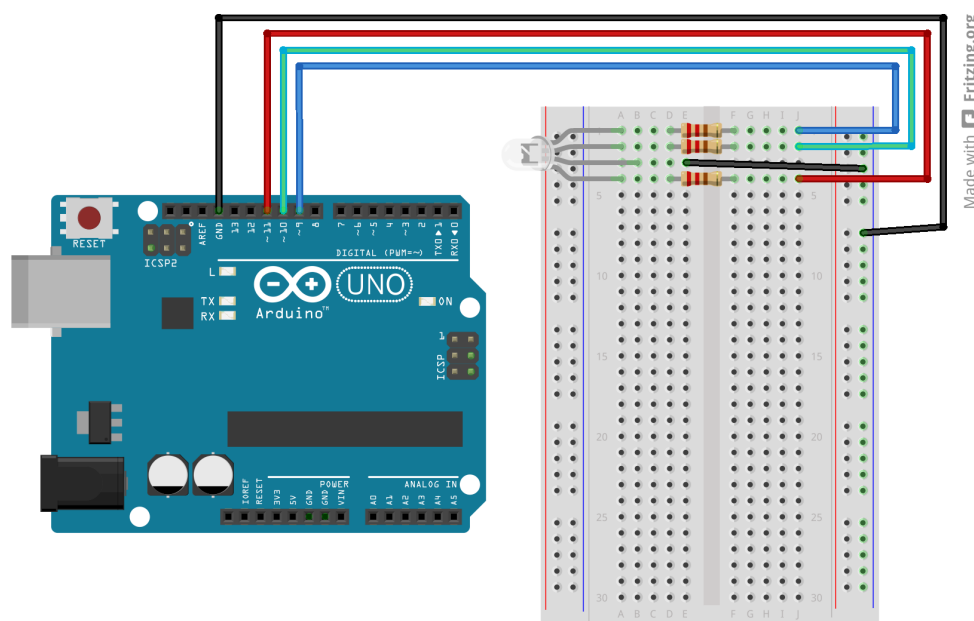
We get the real time zone based time back! This information of RTL functions is contained in various unit files that are a standard part of Delphi or Indy. This collection of units is referred to as the RTL (run time library). The RTL contains large number of functions for you to use.

### 1.1.3 Bread Board Check

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).

With the breadboard you prepared above or below, add three wires for power to the RGB light and one for ground GND for your AVR controller.

Place the 3 resistors and LED as shown. Make sure the longest leg of the LED is to GND, connected to the minus. Resistors don't have a direction, so it doesn't matter which way it goes.



## 5: Breadboard Settings

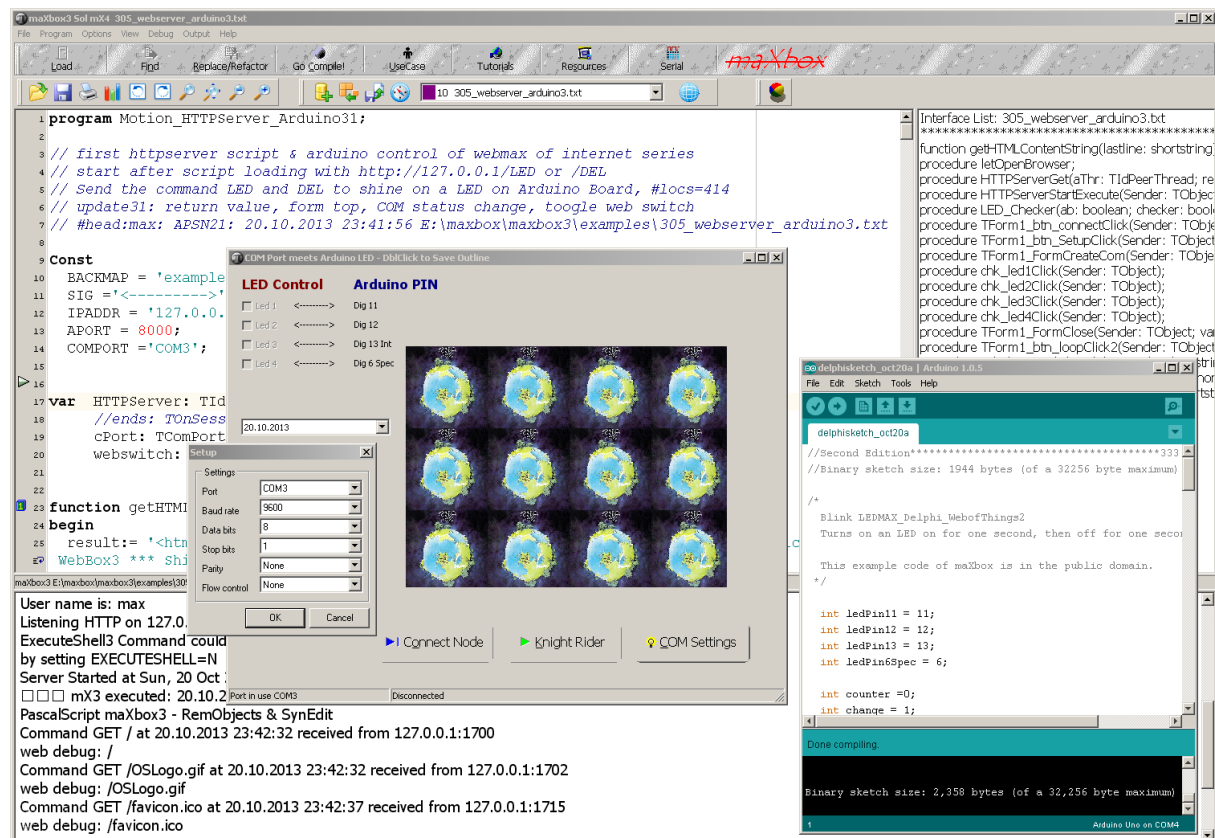
If you're using a standard breadboard, you'll need to use wires to reach the Arduino. Run 3 wires (red, green and blue) to the pin sockets on the Arduino. Run the other wire (black) to one of the GND sockets on the Arduino. The colours aren't essential but they will help you remember what the wires are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

Close to the end of the first part some knowledge about sockets. Socket connections can be divided into three basic types, which reflect how a connection was initiated and what a local socket is connected to. These are

- Client connections.
- Listening connections.
- Server connections.

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 connection is fundamentally different, as it has only a single endpoint.

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.



6: the BOX, Arduino and the GUI

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.

Many of the protocols that control activity on the Internet are defined in Request for Comment (RFC) documents that are created, updated, and maintained by the Internet Engineering Task Force (IETF), the protocol engineering and development arm of the Internet. There are several important RFCs that you will find useful when writing Internet applications:

- RFC822, "Standard for the format of ARPA Internet text messages," describes the structure and content of message headers.
- RFC1521, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies," describes the method used to encapsulate and transport multipart and multifragment messages.
- RFC1945, "Hypertext Transfer Protocol—HTTP/1.0," describes a transfer mechanism used to distribute collaborative hypermedia documents.

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 the const in line 13.



Try to get data back from Arduino as a test case like this:

```
Serial.print() in Arduino and cPort.ReadStr() in maXbox
Function ReadStr(var Str: string; Count: Integer): Integer); //CPort Lib
//S_ShellExecute('http:'+IPADDR+':'+IntToStr(APORT)+'soa_delphi.pdf','',seCmdOpen)
```

~~maXbox~~

## 1.2 QR Code

Now we deal with QR code and put a definition from wiki first:

A QR code (abbreviated from Quick Response Code) is the trademark for a type of matrix bar code (or two-dimensional bar code) first designed for the automotive industry in Japan. The point is that QR codes give users or customers instant access to the information they need on the move; whether its on your website, ask.com or another news channel a QR service is dedicated to making access to it quicker and easier than ever before.

The script you find at the examples:

```
393_QRCode2Direct_detlef.TXT
```

But what's the matter with Web of Things? Simply that each thing, device or gadget will have an identity to communicate, could also be a RFID.

We can put many information (as long as 4296) in a QR code; the code contains information such as a website address, telephone number, device ID or a business card information.



The call is that simple:

```
GetQrCode3 (250,250,'Q',QDATA, ExePath+AFILENAME) ;  
OpenDoc (ExePath+AFILENAME) ;
```

This is so simple because google provides a service we use:

```
UrlGoogleQrCode=  
'http://chart.apis.google.com/chart?chs=%dx%d&cht=qr&chld=%s&chl=%s;
```

Using the Google Chart Tools / Image Charts (aka Chart API) you can easily generate QR codes, this kind of images are a special type of two-dimensional bar-codes. They are also known as hard-links or physical world hyperlinks.

The API requires 4 simple fields be posted to it:

1. cht=qr this tells Google to create a QR code;
2. chld=M the error correction level of the QR code (see here for more info);
3. chs=wxh the width and height of the image to return (e.g. chs=250x250);
4. chl=text the URL encoded text to be inserted into the bar-code.

Some notes about firewalls or proxy-servers when using this service. 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. A firewall that only allows connections to the listening port will also prevent the remote debugger from working. But after all HTTP lets you create clients that can communicate with an application server that is protected by a firewall.

## 1.3 3D Printing

Oh that's amazing the order 1.3 is 3D, just kidding. We just scratch 3D Print topic.  
First the script:

```
maXbox3 356_3D_printer.txt
```

So the world around is big. We are in the middle of a second industrial revolution – 3D printing is about to transform every aspect of our lives, many scientists said. Making a replacement plastic handle should be a tough decision, right?

The script above just shows a simulation of an object to be printed on a Z-Axis.

Remember that the Y-Carriage and the X-Carriage move the heat bed and extruder, respectively. The Z-Axis is not in itself a carriage, holding a small portion of the printer. It holds up the entire X-Axis, and it moves differently than its counterparts. That's all for the print hint ;-)

## 1.4 Web Cam

I made the script with the AVICAP32.DLL aka AVI API:

```
maXbox3 433_video_avicap2.txt
```

I need to get a regular snapshot from a web-cam in my script. Speed is not a problem (once a second is fine enough).

You don't need a real web-cam in the meantime you can running instead a simulator. The simulator works (I can see the video using Skype) but not with the test app.

The `avicap.dll` is used by your computer when capturing video from cameras (e.g. web-cams) to store the resulting video as AVI format. If you are recording video or using a web-cam for video conferencing, you should not stop this process.

These options tell avicap when to shutdown:

- never shut down system  
    avicap will never try to shutdown your system
- shutdown after last recording  
    avicap will shutdown (to poweroff) your system only after the last recording, when there are no pending records  
    you don't need nvram-wakeup for this
- shutdown in-between recordings  
    avicap will shutdown (first to reboot, then to poweroff) your system with nvram-wakeup between recordings

Applications can use video-stream callback functions during streaming capture to process a captured video frame. The capture window calls a video-stream callback function just before writing each captured frame to the disk.

Alternatively, any web-cam demo code would be appreciated, preferably with a known good Exe as well as source.

```
function capCreateCaptureWindowATest(lpszWindowName : pchar;  
    dwStyle: longint;  
    x,y: integer;  
    nWidth,nHeight: integer;  
    ParentWin: HWND;  
    nId: integer): HWND;  
//stdcall external 'AVICAP32.DLL';  
external 'capCreateCaptureWindowA@avicap32.dll stdcall';
```

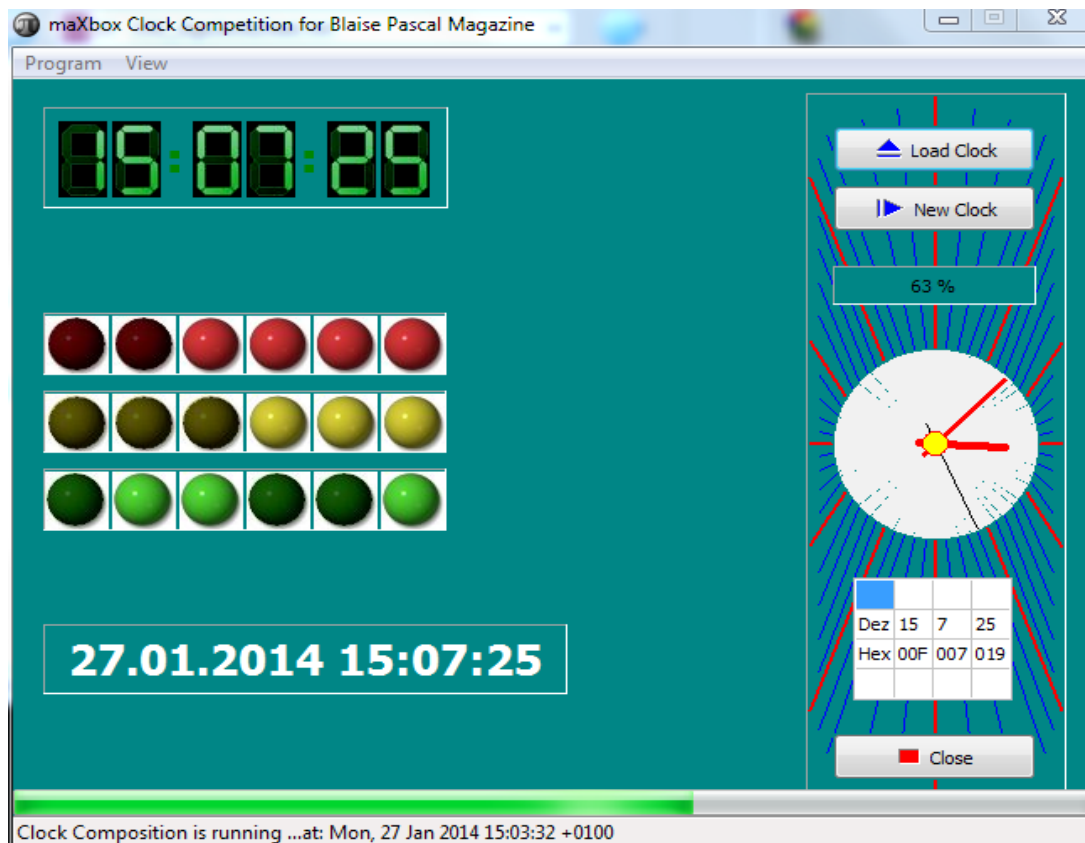
## 1.5 Digi Clock Rock

The next example is about devices. A device study called digit clock.

The script deals with a case study of multi-function digital clocks which provide a reliable and accurate display of time for professional applications with an extensive range of user programmable features and many options including four display colours and five case styles.

The script shows a real wonder of clocks, how many do you count?





7: how many clocks you count?

These digital wall clocks are suitable for stand-alone applications or for synchronized clock system use. All models have the ability to operate as a stopwatch using optional 6 or 18 control panels or rear contacts.

Features lots of customizations, like:

- widget preview during setup
- select widget click actions: tap on widget to load alarm application, widget settings or any installed application
- allows you select your preferred colors for time and date
- shadow effect with selectable color

### 1.5.1 Clock Code

The Code is rather difficult but a timer is always on board:

```

procedure DigiTForm1_TimerTimer(Sender: TObject);
var i: integer; s,s1: String;
    t: TDateTime;
begin
    t:= Now; //Formate, 6-parts of s
    DateTimeToString(s,'hhmmss',t);
    //Number assign and draw
    for i:= 1 to 6 do begin
        s1:= MidStr(s,i,1);
        zeit[i]:= strToInt(s1);
        imgL.Draw(pb[i].Canvas,1,1,zeit[i],true);
    end; //for

```

**end;**

By the way, here's the script:

```
maXbox3 336_digiclock3.txt
```

I use an image list. That is probably the easiest case. Lets draw an image on canvas of some control (`OwnerDraw` situation) and also center it. `X`, `Y` are top left corner of the future image and `Width` is a width of the rectangle we want to center image in.

`MyImageIndex` is an index of the image in the Image List.

Being a little bit more complex, it does require some extra code:

- Make sure that any prior state is cleared.
- If you do not do that with transparency enabled, you would see images on top of each other or scrambled.

## 1.5.2 RTC

For the real time clock, you guess, you need another script and libraries too.

The Arduino has a fairly precise internal clock, but it's not the most well-suited device to keep track of date and time. To get a more accurate date and time with a little backup, we need to use a real-time clock or RTC module. For this we use the well known DS1307 RTC available on a breakout board from SparkFun Electronics /Adafruit Industries, schema in appendix!

The DS1307 is a relatively simple device that uses the I2C bus, or sometimes called Two Wire—although these two terms are not always interchangeable - to send a 7-bit binary number for the current date and time.

The DS1307 Serial Real-Time Clock is a low-power, full binary-coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM. Address and data are transferred serially via a 2-wire, bi-directional bus called I2C.

I2C is an interesting serial protocol that uses two wires, SDA for data and SCL for clock, to control up to 112 slave devices such as sensors, clocks, heart-beats, FM radios, and smart LEDs, from one master device like our micro-controller. In addition to the data and clock pins, each I2C device will also need a positive and ground connection to a power supply rated for a device.

- SCL (Serial Clock Input) – SCL is used to synchronize data movement on the serial interface.
- SDA (Serial Data Input/Output) – SDA is the input/output pin for the 2-wire serial interface. The SDA pin is open drain which requires an external pull-up resistor.

The DS1307 operates as a slave device on the serial bus. Access is obtained by implementing a START condition and providing a device identification code followed by a register address. Subsequent registers can be accessed sequentially until a STOP condition is executed.

## 1.6 Android with Arduino

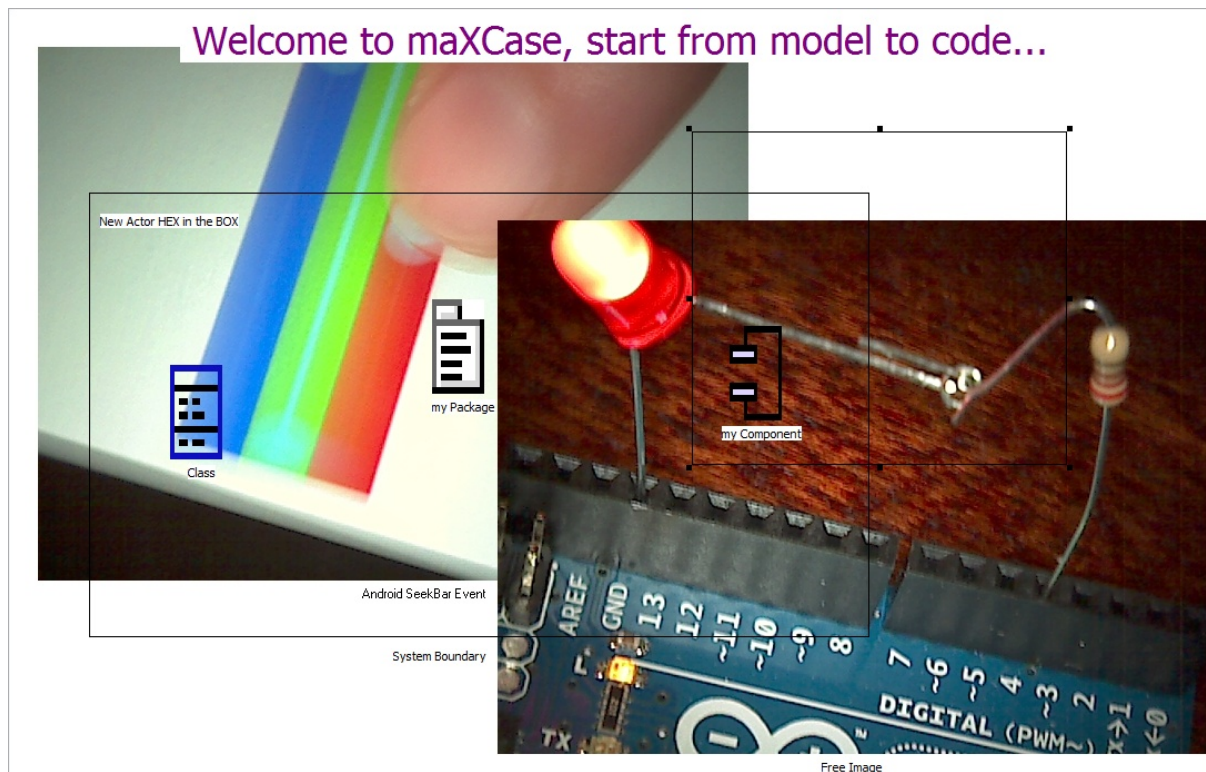
Now at last the dream team with this USB host (I did a lot with my students).

The Arduino ADK is a combination of hardware and software designed to help people interested in designing accessories for Android devices.

During Google's 2011 IO keynote, Google introduced a board based on the Arduino Mega 2560 which includes this USB host. A new USB library [3] was introduced, enabling data to be sent and received from any external devices. This library is included with the latest version of the Android OS. Any devices with OS 3.1 and later can use the USB port to connect to accessories.

<http://labs.arduino.cc/ADK/Index>

App makers can freely use it in their development. This library has been included in 2.3.4, and some devices have been reported to work with 2.3.3. I work with Andro 4.2.2 and Arduino 1.5.



8: A seek bar on Android dims the Red LED on Arduino

The Mega ADK board is a derivative of the Arduino Mega 2560. The modified Mega 2560 board includes a USB host chip. This host chip allows any USB device to connect to the Arduino.

The USB host is not part of the original core of Arduino. To use the new features on this board you will need to include some libraries in your sketches.

There are three libraries needed to make the system RGB SeekBar work:

- MAX3421e: handles the USB host chip
- USB: handles the USB communication
- Android Accessory: checks if the device connecting is one of the available accessory-enabled phones

[http://www.softwareschule.ch/examples/B004\\_RGB\\_Led.ino.htm](http://www.softwareschule.ch/examples/B004_RGB_Led.ino.htm)

```
#include "variant.h"
#include <stdio.h>
#include <adk.h>
#include <Scheduler.h>

#define LED 13
#define RED 2
#define GREEN 3
#define BLUE 4

/* Aware: Serial has to be right below on newline! */
// Accessory descriptor. It's how Arduino identifies itself to Android.
char model[] = "HelloWorldModel"; // model in xml
```

```

char displayName[] = "A Hello World Accessory"; // your Arduino board
char companyName[] = "Hello maXbox Inc";

// Make up anything you want for these
char versionNumber[] = "1.2";
char serialNumber[] = "1";
char url[] = "http://www.osciprime.com/repo/OsciPrimeICS.apk";

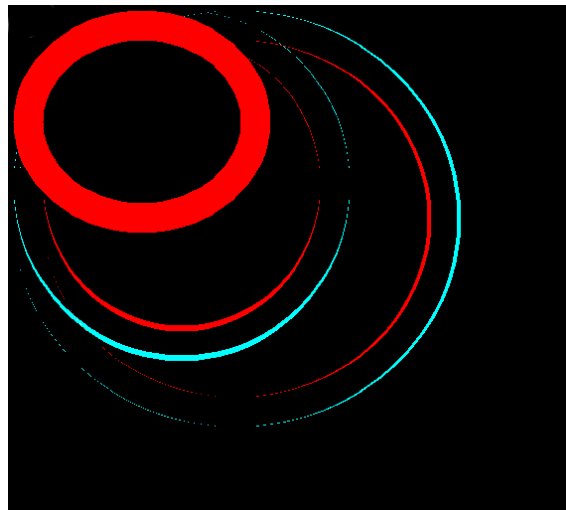
volatile uint8_t pause = 255;
USBHost Usb;
ADK adk(&Usb, companyName, model,
displayName,versionNumber,url,serialNumber);

```

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 as well as a bit of imagination, you can build anything you want. 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, Drones, MIDI Synthesizers, Ethernet connections, etc.) that you can simply slot into the top of your Arduino to get extra functionality.

Arduino is an open-source single-board micro-controller, descendant of the open-source Wiring platform designed to make the process of using electronics in multitude projects more accessible. The Arduino can be connected to Dot matrix displays, glasses, switches, motors, temperature sensors, pressure sensors, distance sensors, web-cams, printers, GPS receivers, Ethernet modules and so on.

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 sensors.



Feedback @  
[max@kleiner.com](mailto:max@kleiner.com)

Literature:

Kleiner et al., Patterns konkret, 2003, Software & Support

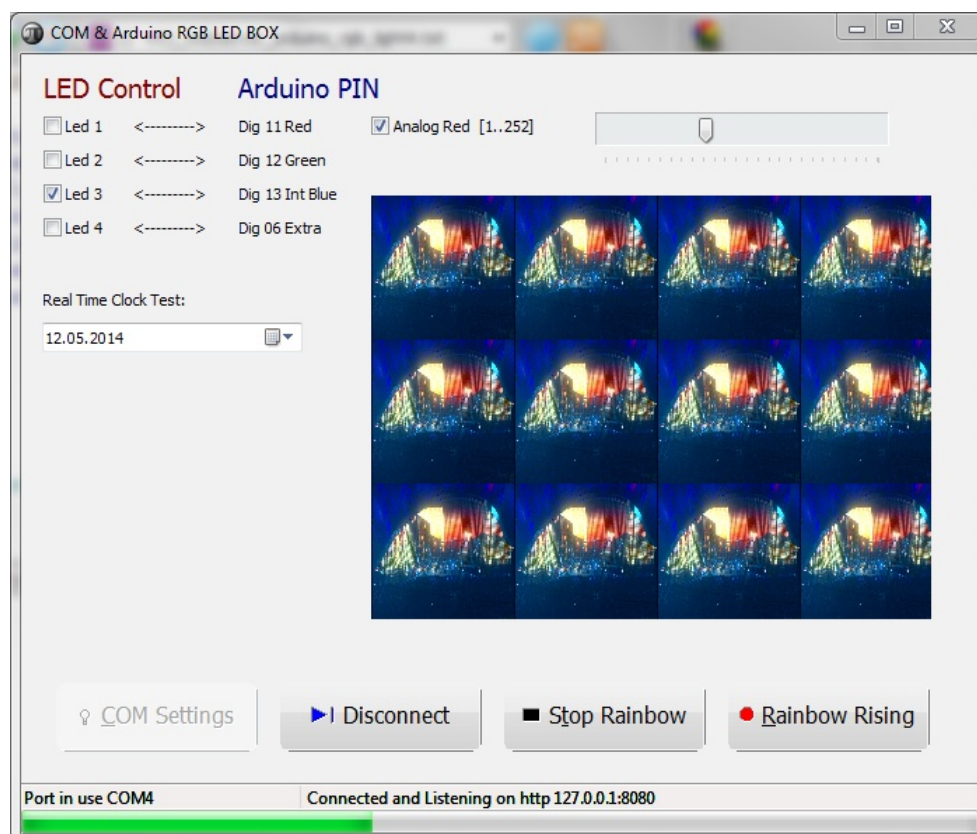
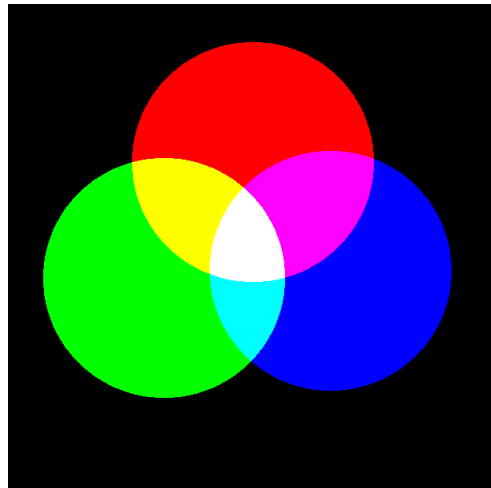
Links of maXbox, Web of Things, Arduino and Indy:

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

[http://www.softwareschule.ch/download/Arduino\\_C\\_2014\\_6\\_basta\\_box.pdf](http://www.softwareschule.ch/download/Arduino_C_2014_6_basta_box.pdf)

<http://www.softwareschule.ch/maxbox.htm>  
<http://www.indyproject.org/Socket/index.EN.aspx>  
[http://www.softwareschule.ch/examples/443\\_webserver\\_arduino\\_rgb\\_light4.htm](http://www.softwareschule.ch/examples/443_webserver_arduino_rgb_light4.htm)  
<http://en.wikipedia.org/wiki/Arduino>  
<http://fritzing.org/>

<http://sourceforge.net/projects/maxbox>  
<http://sourceforge.net/apps/mediawiki/maxbox/>  
<http://sourceforge.net/projects/delphiwebstart>  
[http://www.blaisepascal.eu/subscribers/vogelaar\\_elctronics\\_info\\_english.php](http://www.blaisepascal.eu/subscribers/vogelaar_elctronics_info_english.php)



[maXbox Arduino Framework](#)

## 1.7 Appendix Arduino LED Rainbow

```
procedure TF_Rainbowloop(Sender: TObject);
var mtime, multiple: integer;
begin
    LED_Cheker(false, true);
    {Color Spectrum from Red to White code (r,y,g,c,b,m,w...}
    mtime:= 500; //1000;
    multiple:= 2;
    statBar.Panels[1].Text:='Rainbow - Click LED4 to end loop!';
    try
        with cPort do begin //using
            repeat
                //WriteStr('1'); Sleep(mtime);
                digitalWrite(red, AHIGH); // red
                delay(mtime);
                digitalWrite(green, AHIGH); // yellow
                delay(mtime);
                digitalWrite(red, ALOW); // green
                delay(mtime);
                digitalWrite(blue, AHIGH); // cyan
                delay(mtime);
                digitalWrite(green, ALOW); // blue
                delay(mtime);
                digitalWrite(red, AHIGH); // magenta
                delay(mtime);
                digitalWrite(green, AHIGH); // white
                mtime:= mtime * multiple;
                delay(mtime);
                digitalWrite(blue, ALOW); // reset
                digitalWrite(green, ALOW);
                digitalWrite(red, ALOW);
                mtime:= mtime div multiple; //time/=multiple;
            until chk_led4.Checked=true;
            chk_led4.Checked:= false;
        end;
    except
        Showmessage(R_EXCEPTMESS);
    end;
end;
```

## 1.8 Appendix Arduino Base Code

```
//*****Arduino Code*****
* Turns on and off 5 light emitting diodes (LED) connected to digital pins 2 to 6. The LEDs will be
controlled using check boxes on maXbox that sends serial data to Arduino. */
int val = 0; // variable to store the data from the serial port
int ledPin1 = 2; // LED connected to digital pin 2
int ledPin2 = 3; // LED connected to digital pin 3
int ledPin3 = 4; // LED connected to digital pin 4
```



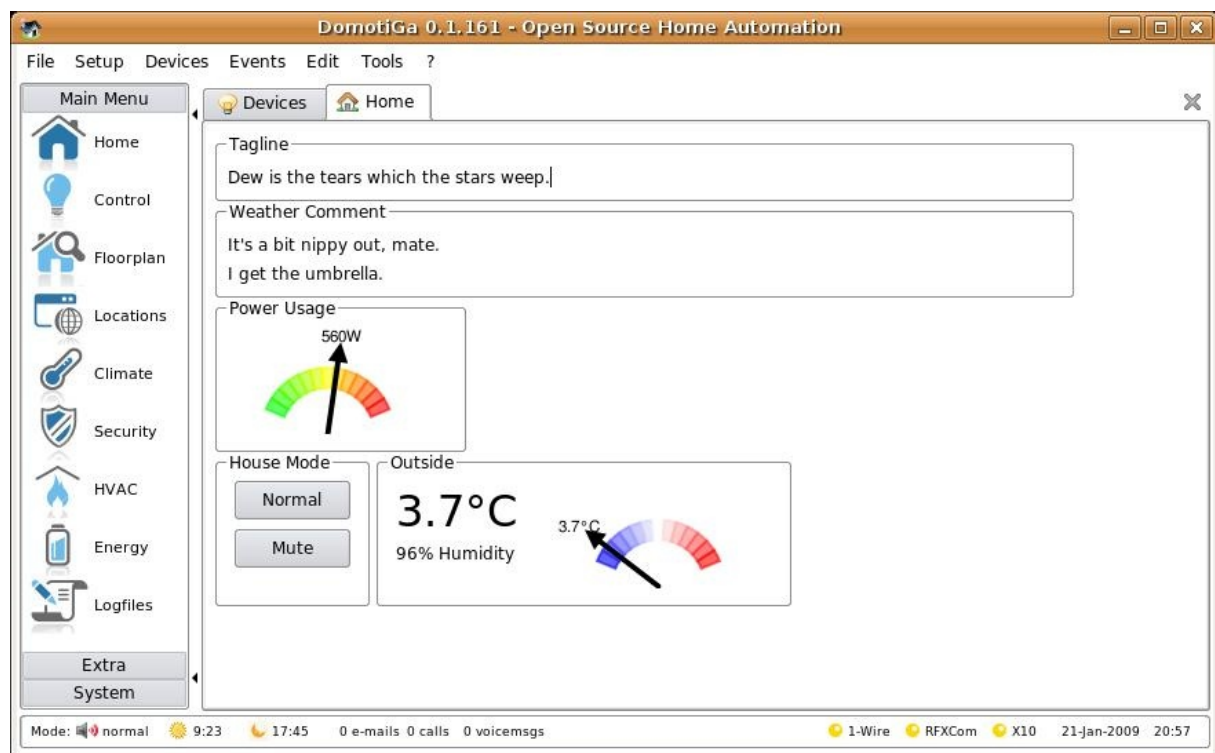
```

void setup() {
  pinMode(ledPin1,OUTPUT); // declare the LED's pin as output
  pinMode(ledPin2,OUTPUT); // declare the LED's pin as output
  pinMode(ledPin3,OUTPUT); // declare the LED's pin as output
  Serial.begin(9600);      // connect to the serial port
}

void loop () {
  val = Serial.read();    // read the serial port
  if (val !=-1){
    if (val=='1'){
      digitalWrite(ledPin1,HIGH);
    }
    else if (val=='A'){
      digitalWrite(ledPin1,LOW);    }
    if (val=='2'){
      digitalWrite(ledPin2,HIGH);    }
    else if (val=='B'){
      digitalWrite(ledPin2,LOW);    }
    if (val=='3'){
      digitalWrite(ledPin3,HIGH);    }
    else if (val=='C'){
      digitalWrite(ledPin3,LOW);    }
    //Serial.println();
  }
}

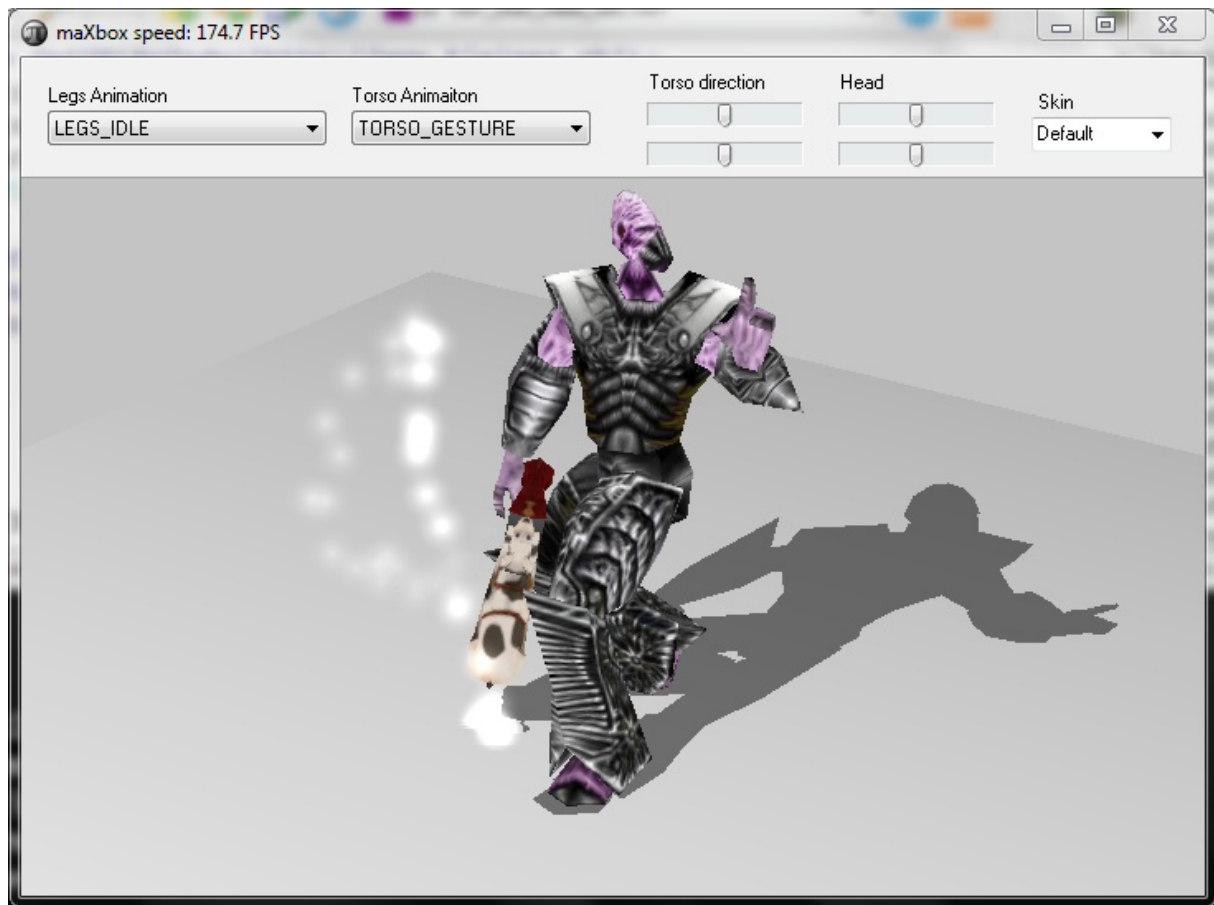
```

## 1.9 Appendix Arduino App

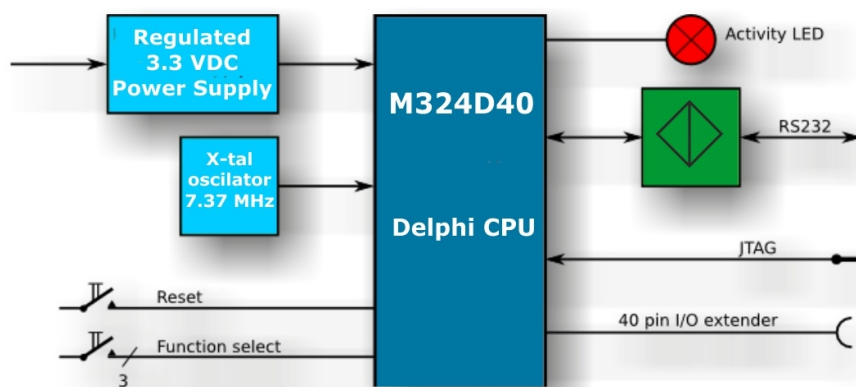




## 1.10 Appendix maXbox Robo App

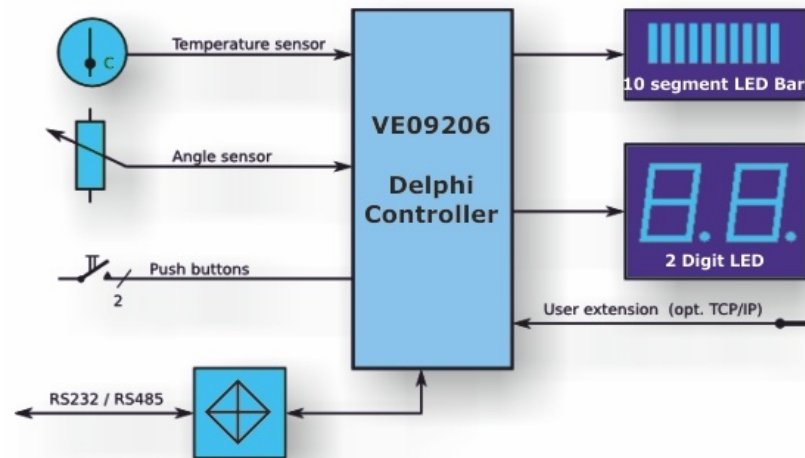


## 1.11 Appendix Delphi Controller

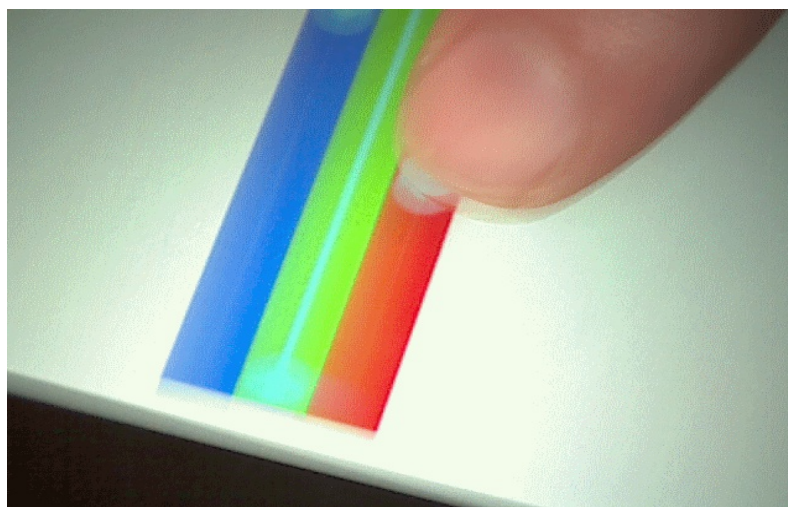
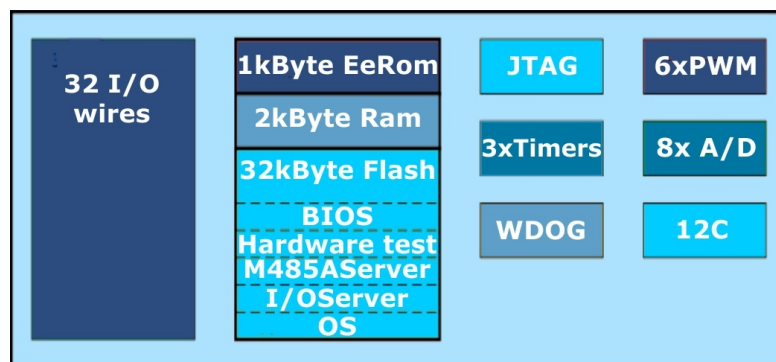


[http://www.blaisepascal.eu/index.php?actie=../subscribers/UK\\_Electronics\\_Department](http://www.blaisepascal.eu/index.php?actie=../subscribers/UK_Electronics_Department)

### 1.11.1 Delphi Schema

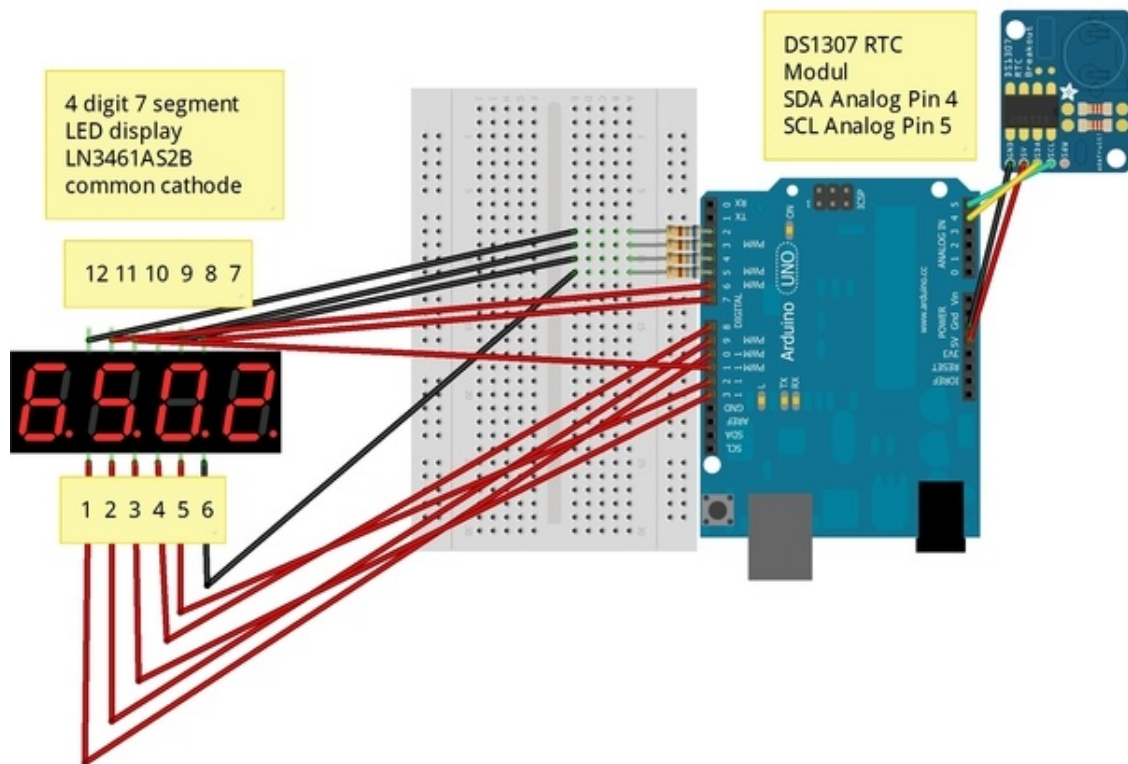


### 1.11.2 Delphi I/O Layout 2



9: maXbox Web Cam live in Android on Arduino

## 1.12 RTC Layout



Made with  Fritzing.org

RTClock:  
,Arduino by Silvia Rothen

<http://www.ecotronics.ch/ecotron/arduinocheatsheet.htm>