

maXbox



maXbox Starter 18

Start with Arduino Programming V2

1.1 Physical Computing

Today we jump into a complete new topic in programming called embedded computing with or without the internet; we code a LED light on the Arduino board which we switch off or on by a browser on an android phone 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/>

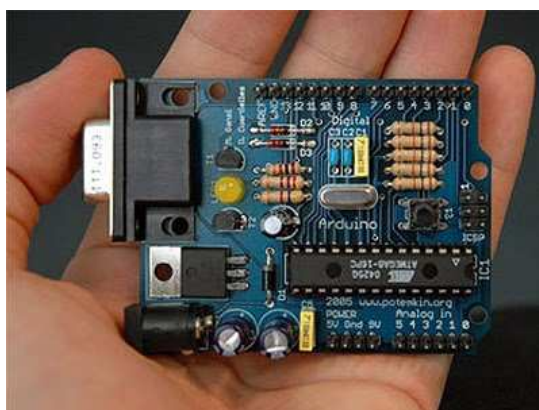
Arduino hardware is programmed using a Wiring-based language (syntax and libraries), similar to C++ and Object Pascal with some slight simplifications and modifications, and a Processing-based integrated development environment like Delphi.

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.

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 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 communication with the microcontroller of Arduino¹.




¹ An Arduino board consists of an 8-bit Atmel AVR [microcontroller](#)

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 localhost. It is common for computers to refer to themselves with the name localhost and the IP number 127.0.0.1.

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

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 the browser too.

 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:

```
305_webserver_arduino3.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 or direct as a file
http://www.softwareschule.ch/examples/305_webserver_arduino3.txt

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

```
01 program Motion_HTTPServer_Arduino3;
```

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

 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 Crnila² 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 application easier and faster than ever.

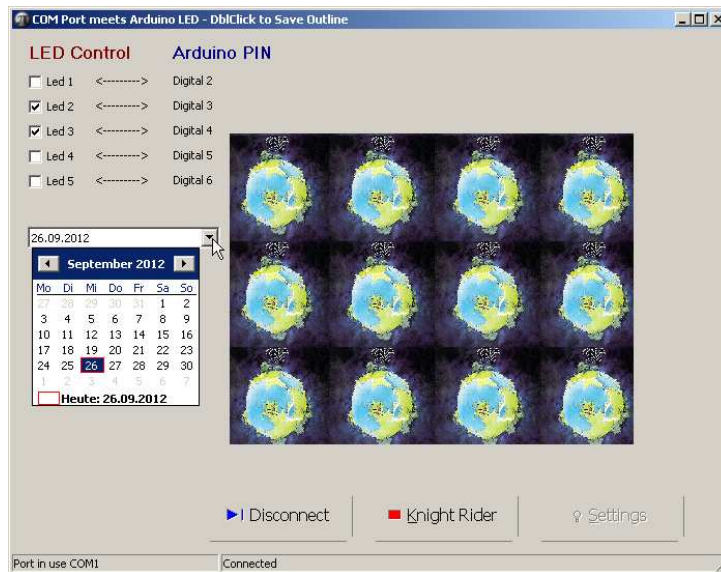
First we start with the web server and second we explain the COM port.

After creating the object in line 89 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.

```
89 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; another advantage of modular programming. Once a unit is tested it's stable to use.

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



1: The GUI of the App

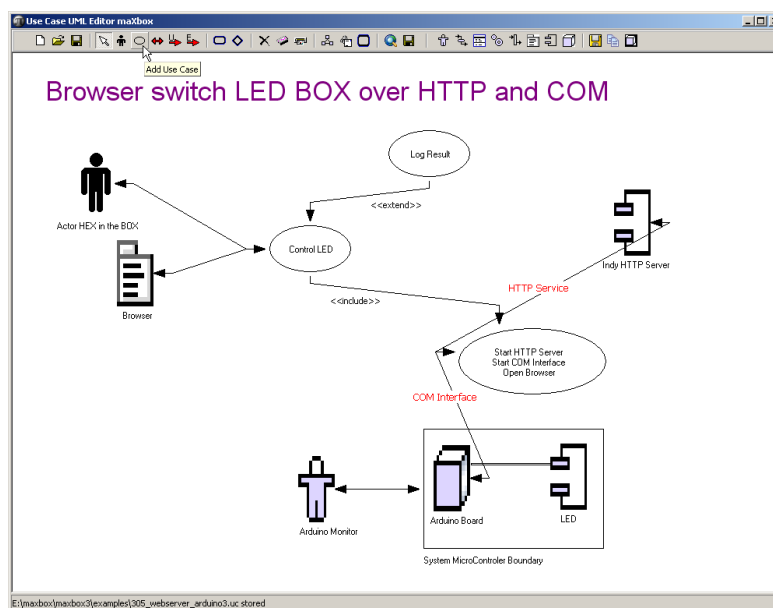
Indy is designed to provide a very high level of abstraction. Much more stuff or intricacies and details of the TCP/IP stack are hidden from the Indy programmer. A typical Indy client session looks like this:
with IndyClient **do begin**

```
Host:= 'zip.pbe.com'; // Host to call
Port:= 6000; // Port to call the server on
Connect; // get something to 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.



2: The UC of the App

So let's get back to our HTTP Create in line 89. In line 95 and 96 you see the port and IP address configuration of a `const` in line 12, instead of IP you can also set a host name as a parameter.

```
90 with HTTPServer do begin
91   if Active then Free;
92   if not Active then begin
93     Bindings.Clear;
94     bindings.Add;
95     bindings.items[0].Port:= APORT;
96     bindings.items[0].IP:= IPADDR; //'127.0.0.1';
97     Active:= true;
98     onCommandGet:= @HTTPServerGet;
99     Printf('Listening HTTP on %s:%d.', [Bindings[0].IP, Bindings[0].Port]);
100    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`.

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 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 to switch off is `../DEL (127.0.0.1:8000/LED)`. Those are GET commands send with the browser.

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: OK. Its time for lord of the things magix:

```
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;
```

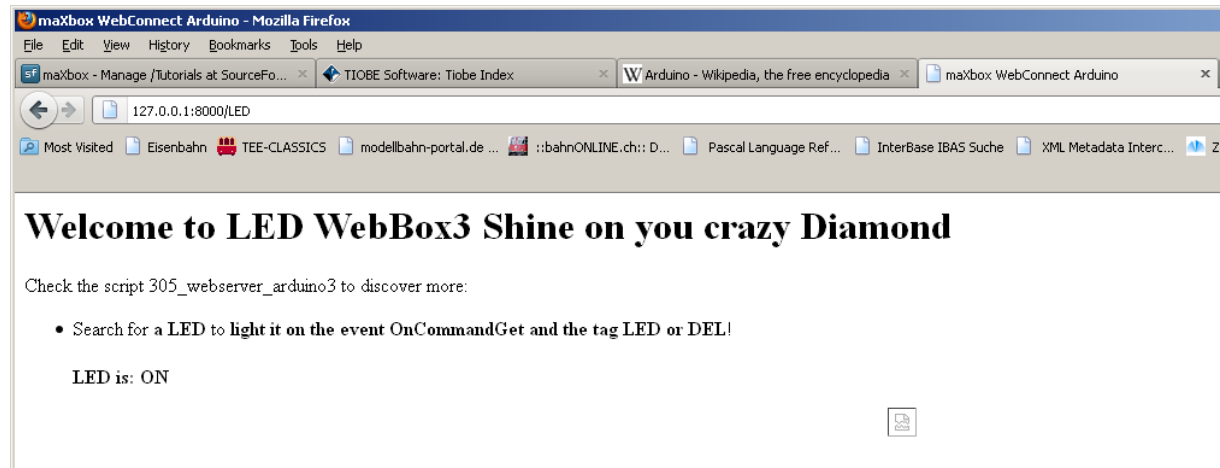
HTTP request messages contain many headers that describe information about the client, the target of the request, the way the request should be handled, and any content sent with the request. Each header is identified by a name, such as "Host" followed by a string value.

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 composes a request.

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

```
69  webswitch:= NOT webswitch;
70  if webswitch then begin
71    cPort.WriteStr('1')
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
```

The acronym HTML stands for Hyper Text Markup Language - the primary programming language used to write content on the web. 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 the extension html. Or you copy the output and paste it in a new maXbox instance. Then you click on the right mouse click (context menu) and change to HTML Syntax!



3: The Output Window


When the browser starts from the script the server is ready for commands to pass as chars in line 55 or 60 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.

```
60  writeln(localcom+ ' : LED on');
61  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 the syntax before run is F2 or the **Syntax Check** in the menu

Program. When you run this code from the script `102_pas_http_download.txt` you will see the 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
end
```

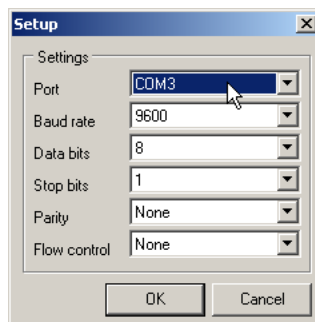
 The Object `TIdHTTP` is a dynamically allocated block of memory whose structure is determined by its class type. Each object has a unique copy of every field defined in the class, but all instances of a class share the same methods. With the method `Get1` you can download files.

```
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 in line 40, it's the event `onCommandGet` with the corresponding event handler method `@HTTPServerGet()` and one object of `TIdPeerThread`. You can use them at server as the way to serve files of many kinds!



4: COM Port Settings

1.3 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 in which COM Port the USB Hub works) or try it with the app in picture 4 on the button "Settings":

```
procedure TForm1_FormCreateCom(Sender: TObject);
begin
  cPort:= TComPort.Create(self);
```

```

with cPort do begin
    BaudRate:= br9600;
    Port:= COMPORT; //'COM5';
    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 the code site to the Arduino Editor to explain how he handles our commands (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 the data from the serial port
int ledPin11 = 11;    // LED connected to digital pin 11 or the inbuilt 13!

void setup() {

    pinMode(ledPin11,OUTPUT); // declare the LED's pin as output

    Serial.begin(9600);      // connect to the 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 it is 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 Serial Monitor window or in maXbox.

In Line 412 we find a last function of the RTL (Runtime Library) of Indy:

```

412 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 a very large number of functions and procedures for you to use.

By the way: In my research and in my debugging, I found that the function `GetTimeZoneInformation` was returning a value oriented towards converting a Time from GMT to Local Time. We wanted to do the reverse for getting the difference. The issue with `TidMessage.UseNowForTime = False` bug was that the `TidMessage` was calling Borland's date function instead of using the `Date` property, see Appendix.

1.4 FrameworkFlow

At last but not least some words about sockets and streams. A listening server socket component automatically accepts client connection requests when they are received. You receive notification every time this occurs in an `OnCommandGet` event.

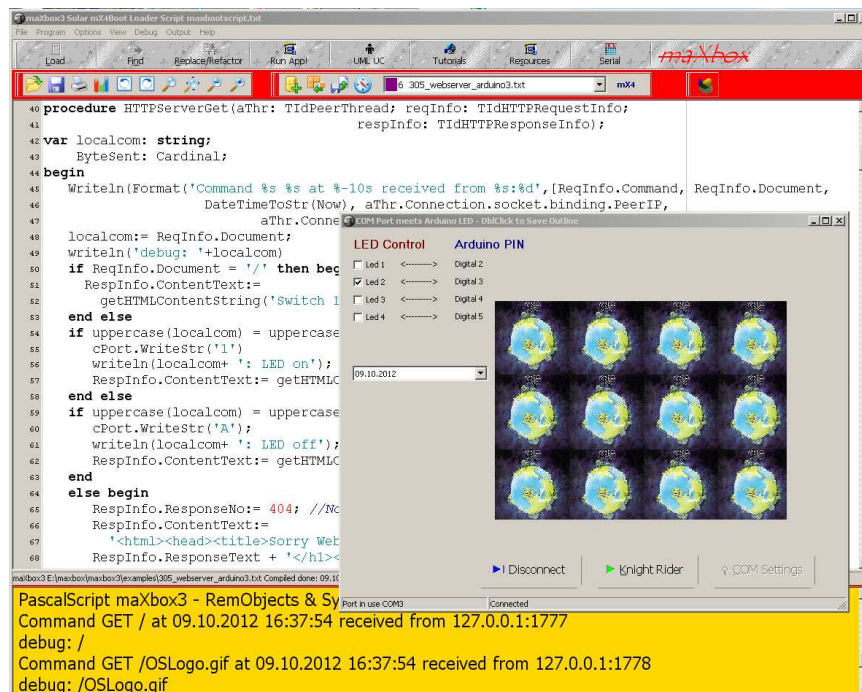
Server connections are formed by server sockets when a listening socket accepts a client request. A description of the server socket that completes the connection to the client is sent to the client when the server accepts the connection. The connection is then established when the client socket receives this description and completes the connection.

Socket connections can be divided into three basic types, which reflect how the connection was initiated and what the 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 the interface between your network server or client application and the networking software. You must provide the interface between your application and the clients that use it. You can copy the API of a standard third party server (such as Apache), or you can design and publish your own API.



5: the BOX 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 68 of the `IP:= IPADDR` with a DHCP or dynDNS address, so you can reach Arduino from an Android App, but change also the const in line 10.



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

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. 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.

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 as well as a bit of imagination, you can build anything you want.

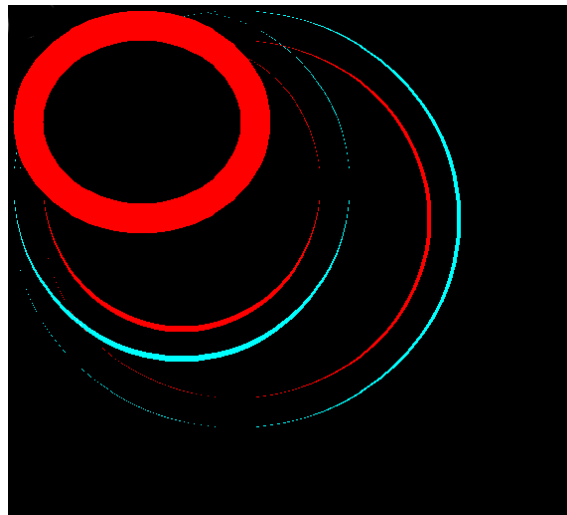
The Arduino can also be extended with the use of Shields which circuit boards are containing other devices (e.g. GPS receivers, LED Cubes, LCD Displays, Sneakers, MIDI Synthesizers, Ethernet connections, etc.) that you can simply slot into the top of your Arduino to get extra functionality.

Next *Tutorial Nr. 19* shows the topic “More Physical Computing” with the Arduino Board and Android.

Arduino is an open-source single-board microcontroller, 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 LED's, Dot Matrix displays, LED displays, buttons, switches, motors, temperature sensors, pressure sensors, distance sensors, webcams, 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 microcontroller 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 microcontrollers 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/maxbox.htm>

<http://www.indyproject.org/sockets/index.EN.aspx>

<http://en.wikipedia.org/wiki/Arduino>

<http://sourceforge.net/projects/maxbox>

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

<http://sourceforge.net/projects/delphiwebstart>

1.5 Appendix

```
Function DateTimeToInternetStr(const Value: TDateTime): String;
var
  strOldFormat, strOldTFormat,
  strDate: String;
  wDay,
  wMonth,
  wYear:WORD;
begin
  {needed to prevent wild results}
  Result := '';
  strOldFormat := ShortDateFormat ;

  ShortDateFormat := 'DD.MM.YYYY';

  // Date
  case DayOfWeek(Value) of
    1: strDate := 'Sun, ';
    2: strDate := 'Mon, ';
    3: strDate := 'Tue, ';
    4: strDate := 'Wed, ';
    5: strDate := 'Thu, ';
    6: strDate := 'Fri, ';
    7: strDate := 'Sat, ';
  end;
  DecodeDate(Value, wYear, wMonth, wDay);
  strDate := strDate + IntToStr(wDay) + #32;
  case wMonth of
    1: strDate := strDate + 'Jan ';
    2: strDate := strDate + 'Feb ';
    3: strDate := strDate + 'Mar ';
    4: strDate := strDate + 'Apr ';
    5: strDate := strDate + 'May ';
    6: strDate := strDate + 'Jun ';
    7: strDate := strDate + 'Jul ';
    8: strDate := strDate + 'Aug ';
    9: strDate := strDate + 'Sep ';
    10: strDate := strDate + 'Oct ';
    11: strDate := strDate + 'Nov ';
    12: strDate := strDate + 'Dec ';
  end;
  //Correction
  strOldTFormat := LongTimeFormat;
  LongTimeFormat := 'HH:NN:SS';
  strDate := strDate + IntToStr(wYear) + #32 + TimeToStr(Value);
  Result := strDate + #32 + DateTimeToGmtOffsetStr(OffsetFromUTC,False);
  LongTimeFormat := strOldTFormat;
{
  strOldTFormat := LongDateFormat;
  LongDateFormat := 'HH:NN:SS';
  strDate := strDate + IntToStr(wYear) + #32 + TimeToStr(Value);
  LongDateFormat := strOldTFormat;
  Result := strDate + #32 + DateTimeToGmtOffsetStr(OffsetFromUTC,False);
  ShortDateFormat := strOldFormat ;
}
end;
```

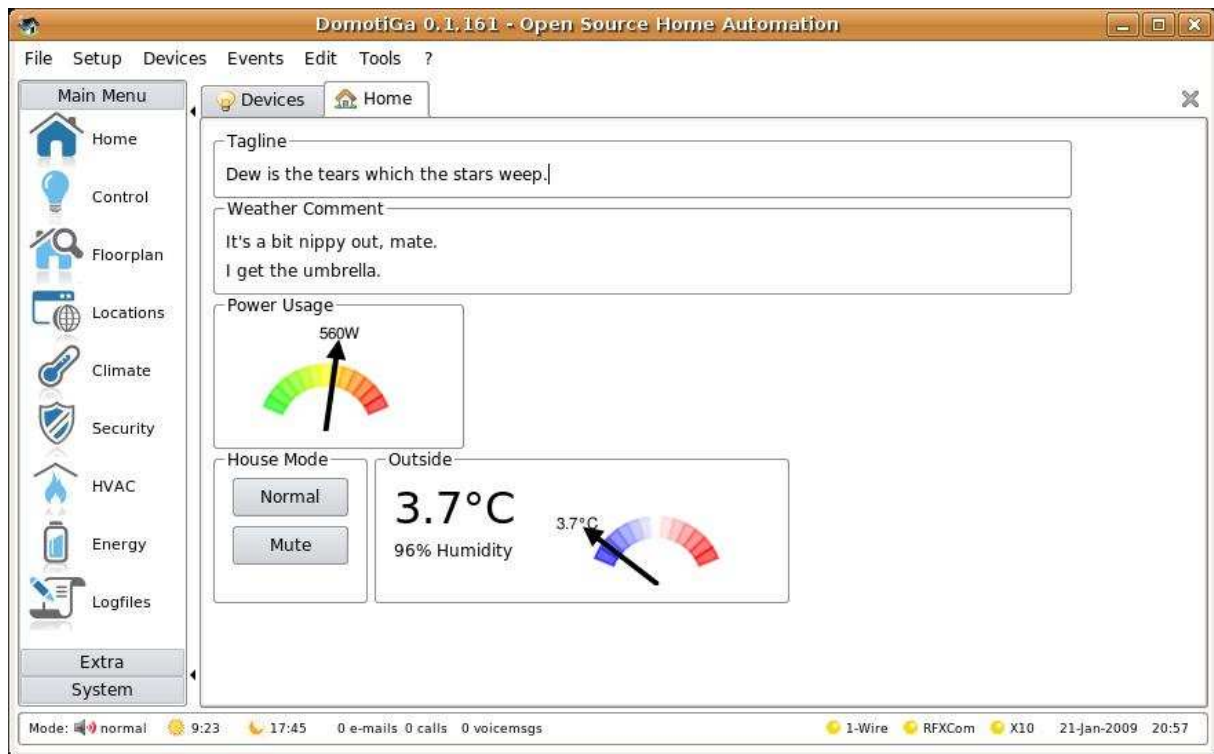
1.6 Appendix Arduino Code

```
/**
 * Delphi LEDs Control
 * -----
 * Turns on and off 5 light emitting diodes(LED) connected to digital
 * pins 2 thru 6. The LEDs will be controlled using check boxes on maXbox that sends serial data to Arduino.
 * IMPORTANT: Don't forget to download the Example 305_webserver_arduino3.txt that controls the leds
 * connected to arduino board in a browser or GUI.
 *
 * Created April 02 2009
 * copleft 2009 Roberto Ramirez <beta@thepenguincult.com>
 * Full Source code at http://www.thepenguincult.com/proyectos/arduino-delphi-control/
 */
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

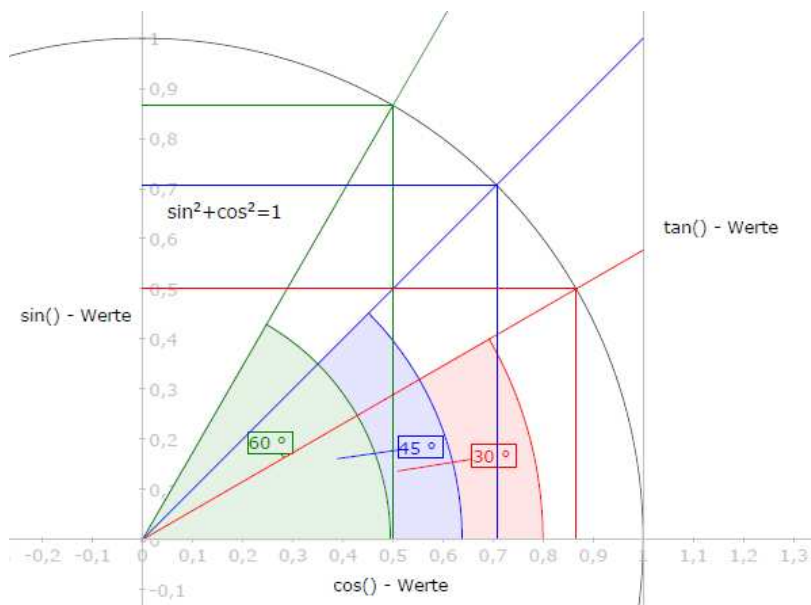
void setup() {
  pinMode(ledPin1,OUTPUT); // declare the LED's pin as output
  pinMode(ledPin2,OUTPUT); // declare the LED's pin as output
  pinMode(ledPin3,OUTPUT); // declare the LED's pin as output
  Serial.begin(9600); // connect to the serial port
}

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();
  }
}
```

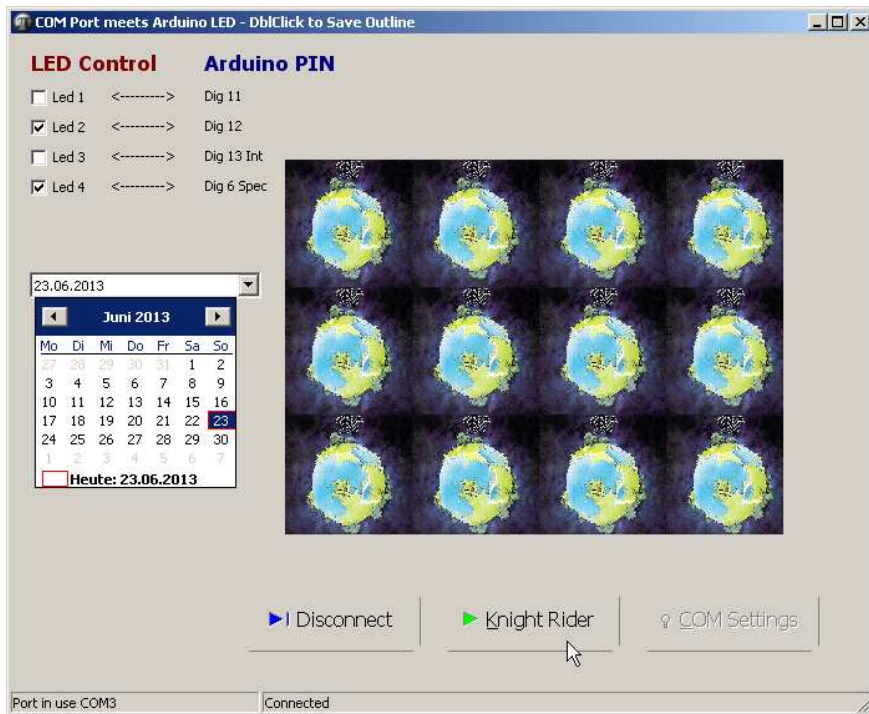
1.7 Appendix Arduino App



1.8 Appendix maXbox App



1.9 Appendix maXbox Arduino App



1.10 Web of Logic Things

