

maXbox



maXbox Starter 17

Start with Web Server Programming

1.1 Second Step of Indy

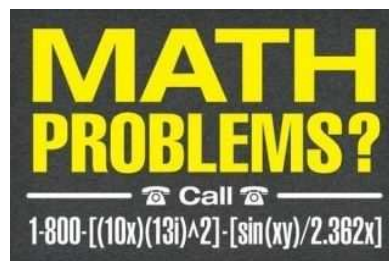
Today we spend another time in programming with the internet, we build our own web server and their protocols too. Hope you did already work with the Starter 1 till 16 at:

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

This lesson will introduce you to Indy Sockets with the TCP-Server, HTTP-Server components and the library in a separate way. So what are the Indy Sockets?

Indy.Sockets (VCL) is an open source socket library that supports clients, servers, TCP, UDP, raw sockets, as well as over 100 higher level protocols such as SMTP, POP3, FTP, HTTP, and many more. Indy.Sockets is written in Delphi but is available for C#, C++, Delphi any .net language, and Kylix (CLX) too, hope you don't have math problems ;-).


In our case we show one example of a HTTP server (and a second one at your own service).



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:

```
303_webserver_alldocs2.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/303_webserver_alldocs2.txt

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

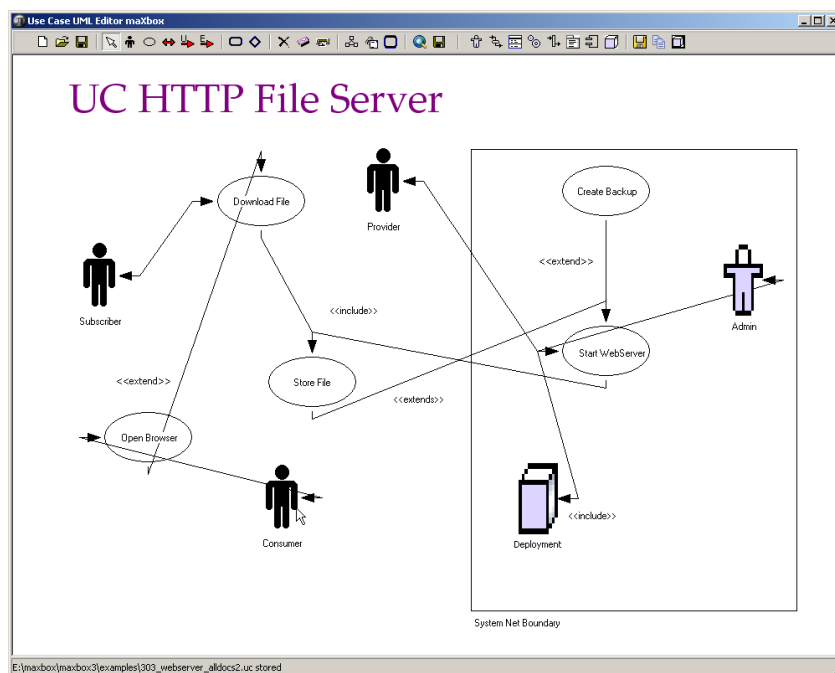
```
01 program Motion_HTTPServer_ALLDocuments2;
```

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

👉 This example requires two objects from the classes: `TIdCustomHTTPServer` and `TIdMIMETable` so the second one is just for converting of the MIME type (see below). After creating the object in line 60 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.

```
60 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 for many times in different projects and also hide difficult sections of code from the developer; another advantage of modular programming. Once a unit is tested it's stable to use.



1: The UC of the Code

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
  Disconnect;
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, using this technique is little complicated to develop a protocol. If your protocol has many commands, your code will be very unintelligible.

By the way almost all server components are inherited from the class `TIdTCPServer` (there are some exception).

So let's get back to our HTTP Create in line 60. In line 66 and 67 you see the port and IP address configuration of a `const` in line 6, instead of IP you can also set a host name as a parameter.

```

61  with HTTPServer do begin
62    if Active then Free;
63    if not Active then begin
64      Bindings.Clear;
65      bindings.Add;
66      bindings.items[0].Port:= APORT;
67      bindings.items[0].IP:= IPADDR; //'127.0.0.1';
68      Active:= true;
69      onCommandGet:= @HTTPServerGet;
70      Printf('Listening HTTP on %s:%d.', [Bindings[0].IP, Bindings[0].Port]);
71    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 in 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 something 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 now what's MIME (Multipurpose Internet Mail Extensions)? For example, you must associate filename extensions—such as `txt`, `.html`, `.pdf`, `.zip` and `.xml`—with MIME types and have the factory or a function to load this data into the control. The following function shows the magix behind:

```

14 function GetMIMEMType(sFile: TFileName): string;
15 var aMIMEMap: TIdMIMETable;
16 begin
17     aMIMEMap:= TIdMIMETable.Create(true);
18     try
19         result:= aMIMEMap.GetFilesMIMEMType(sFile);
20     finally
21         aMIMEMap.Free;
22     end;
23 end;

```

TIdMIMETable provides a generic interface for creating any MIME type as the content of an HTTP response message. Its descendants include page producers and table producers: Without this function the server will in fact deliver but no conversion will be made.

```

36 RespInf.ContentType:= GetMIMEMType(LocalDoc);
37 if FileExists(localDoc) then begin
38     ByteSent:= HTTPServer.ServeFile(AThr, RespInf, LocalDoc);

```

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!

```

maXbox D:\franktech\maXbox\maXbox2\maXbox2_9\maXbox2\mx292\101_pas_http_tester.txt Compiled done: 28.07.10 17:36:29
1 :      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2 : <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de" lang="de" dir="ltr">
3 :     <head>
4 :         <title>softwareschule.ch</title>
5 : <LINK REL="SHORTCUT ICON" HREF="max.ico">
6 :     <!-- Meta-Informationen -->
7 :         <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
8 :         <meta http-equiv="Content-Script-Type" content="text/javascript" />
9 :         <meta http-equiv="Content-Style-Type" content="text/css" />
10 :
runNMax executed on: 28.07.10 17:36:30
PascalScript maXbox; (c) 2005 by RemObjects & SynEdit

```

2: The Output Window

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://www.softwareschule.ch')
        for i:= 1 to 10 do
            memo2.lines.add(IntToStr(i)+' :'+memo2.lines[i])
            //idhttp.get2('http://www.softwareschule.ch/maxbox.htm')
        finally
            idHTTP.Free
        end
    end
end

```

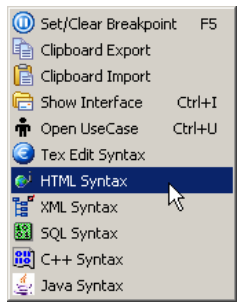
☞ The Object `TIdHTTP` is a dynamically allocated block of memory whose structure is determined by its class type. Each object has a unique copy of every field defined in the class, but all instances of a class share the same methods. With the method `Get1` you can download files.

```
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 69, 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!



1.3 Serve the Command

Using sockets, you can write network servers or client applications that read from and write to other systems. A server or client application is usually dedicated to a single service such as Hypertext Transfer Protocol (HTTP) or File Transfer Protocol (FTP). Using server sockets, an application that provides one of these services can link to client applications that want to use that service. The TCP or HTTP-server component comes from Indy which has two great advantages: "CommandHandlers" and "IDThreads" are a collection of text commands and commands that will be processed by the server. This property greatly simplifies the process of building servers based on text protocols with threads.

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 though a parameter:

```
26 procedure HTTPServerGet(aThr: TIdPeerThread; reqInf: TIdHTTPRequestInfo;
27                           respInf: TIdHTTPResponseInfo);
```

Or in a TCP socket:

```
IdTCPServer1Execute(AThread: TIdPeerThread)
```

When our server is started, this thread transfers all the communication operations to another thread. This technique is very efficient because your client application will be able to connect any time, even if there are many different connections to the server.

```
35   localDoc:= ExpandFilename(Exepath+ '/web'+ReqInf.Document);
36   RespInf.ContentType:= GetMIMETYPE(LocalDoc);
37   if FileExists(localDoc) then begin
38     ByteSent:= HTTPServer.ServeFile(AThr, RespInf, LocalDoc);
```

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.

In line 38 is the heart of the server the method `ServeFile()`. This function with a `TThread` as parameter checks if a `Content-Type` was specified and if not, it detects automatically the `Content-Type` as `MIME`. `TIdHTTPServer` allows the user to assign a custom `Content-Type` before calling `ServeFile()`. `ServeFile` uses `MIMETable` to determine the content type for the requested file and uses `AResponseInfo` to write the HTTP response headers, and writes the content of the file in the parameter to the peer thread `Connection`.

```
Function ServeFile(AThread: TIdPeerThread; ResponseInfo: TIdHTTPResponseInfo;
                  aFile: TFileName): cardinal;
```

Parameters:


- `AThread: TIdPeerThread`
 - The peer thread requesting the file.
- `ResponseInfo: TIdHTTPResponseInfo`
 - The response object used to write headers for the HTTP response.
- `aFile: TFileName`
 - The file name requested.
- Return Value
 - `Cardinal` - Number of bytes written to the peer thread connection.

<http://kumanov.com/docs/prog/indy/007271.html>

The problem was that I did not change the `Content-Type` and it by default was `text/HTML` like line 31 in the code remarks. Changing the content-type made the client write directly to the stream.

Streams are classes that let you read and write data. They provide a common interface for reading and writing to different media such as memory, strings, sockets, and `BLOB` fields in databases. There are several stream classes, which all descend from `TStream`.

`TFileStream` reads from or writes to a file.

 So we pass to `Get1` the host name and a file stream. After writing to the file you can open the zip. Objects are created and destroyed by special methods called constructors and destructors.

The constructor:

```
zipStream:= TFileStream.Create();
```

The destructor (just the free method that calls the destructor):

```
zipStream.Free;
```

In Line 83 we find a last function of the RTL (Runtime Library) of Indy:

```
83 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

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.

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 ☺

```
51 procedure letOpenBrowser;  
52 // TS_ShellExecuteCmd = (seCmdOpen,seCmdPrint,seCmdExplore);  
53 begin  
54  
//ShellAPI.ShellExecute(Handle,PChar('open'),'http://127.0.0.1:8000/',Nil,Nil,0);  
55 S_ShellExecute('http:'+IPADDR+':'+IntToStr(APORT)+' /index.htm', '',seCmdOpen)  
56 end;
```



Try to change the IP address in line 68 of the `IP:= IPADDR` with a DHCP address, so you can download and save files from other clients too, but change also the const in line 6.



Try to download a *.pdf file as a test case like this:

```
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 topic of the web server in an internet.

Next Tutorial Nr. 18 shows the topic "Physical Computing" with the Arduino Board.

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.



Feedback @

max@kleiner.com

Literature:

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

Links of maXbox and Indy:

<http://www.softwareschule.ch/maxbox.htm>

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

<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;
```