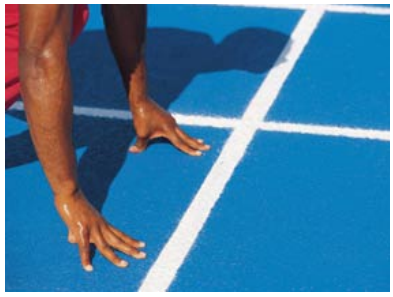
maXbox



# maXbox Starter 71

## From CGI to WebSocket

### 1.1 Common Gateway Interface

CGI is a Common Gateway Interface. As the name says, it is a "common" gateway interface for everything. Quite simply, CGI stands for ***Common Gateway Interface***. That's a fancy term for something we all know as Application Programming Interface. So, CGI is the **API** for the web server. It is so trivial and naïve from the name, but lets start with a first script:

```
program CGI1;

{$mode objfpc}{$H+}
{$APPTYPE CONSOLE}

uses
  {$IFDEF UNIX}{$IFDEF UseCThreads}
  cthreads,
  {$ENDIF}{$ENDIF}
  Classes, SysUtils, CustApp   { you can add units after this };
  begin
   writeln('content-type: text/html');
   writeln('');
   writeln('<html>');
   writeln('<head>');
   writeln('<title>HTML CGI Demo</title>');
   writeln('<meta http-equiv="refresh" content="5">');
   writeln('</head>');
   writeln('<body>');
   writeln('*******************************************************************************');
   writeln(' <h1>Welcome to OpenSSL & maXbox4 CGI scripts</h1>');
   writeln('*******************************************************************************');
   writeln('<br>');
   writeln('Hello, CGI maXbox world!');
```

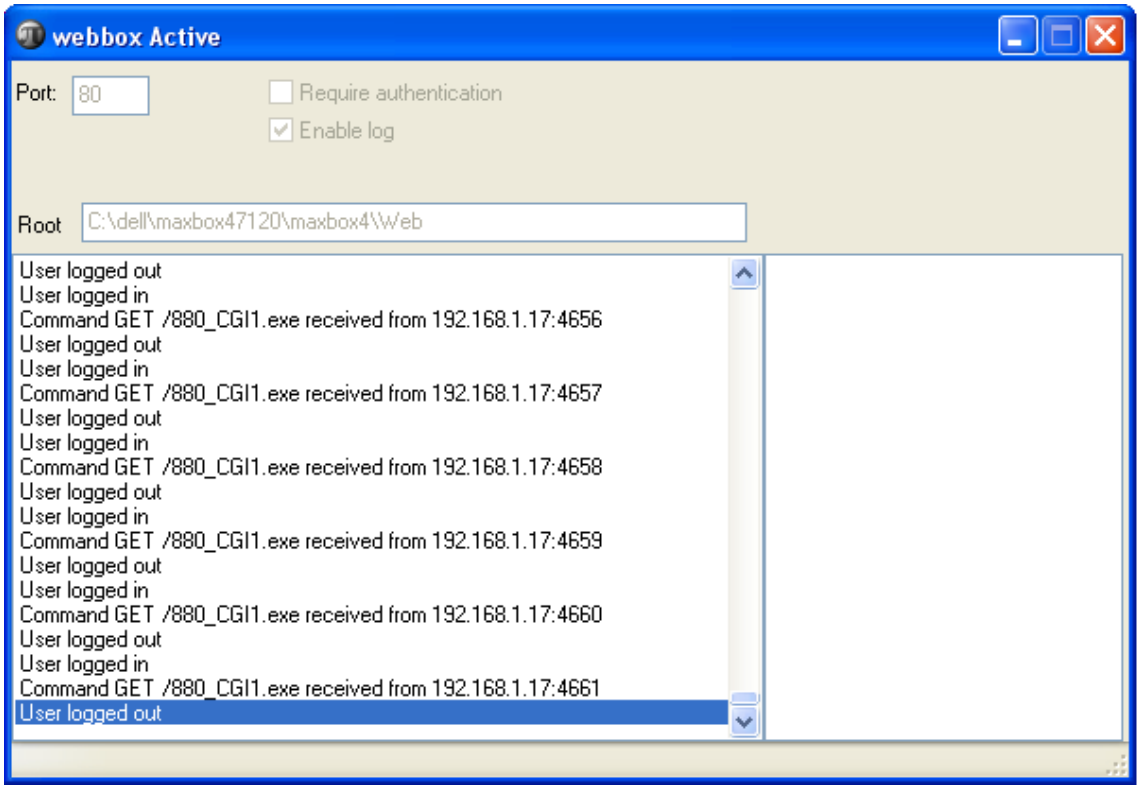
```
  writeln('<br>');
  writeln(dateTimetoStr(now));
  writeln('</body>');
  writeln('</html>')
end.
//end.
```

The interesting one in above script is the calling of a compiled function `writeln(dateTimetoStr(now));` which is an internal function of your language, so you don't need Java Script!

Remember also: The server and the client (the browser) usually run on different computers. They may run under different operating systems, even with different microprocessors. The browser really asks for a "resource" and does not know, or care, where the server gets the data from. First thing we have to do is to compile the script above .

Then we put the exe in the cgi_bin directory or configure in apache:

```
<form action="C:\WWW\cgi-bin\CGI1.exe " method="get">
```




As you see I did a refresh in the CGI script so we can see every 5 seconds a GET -request received from:

```
writeln('<meta http-equiv="refresh" content="5">');
```

CGI and ISAPI server-side applications communicate using HTTP, which is a state-less protocol. This means that in order to save "state-information",

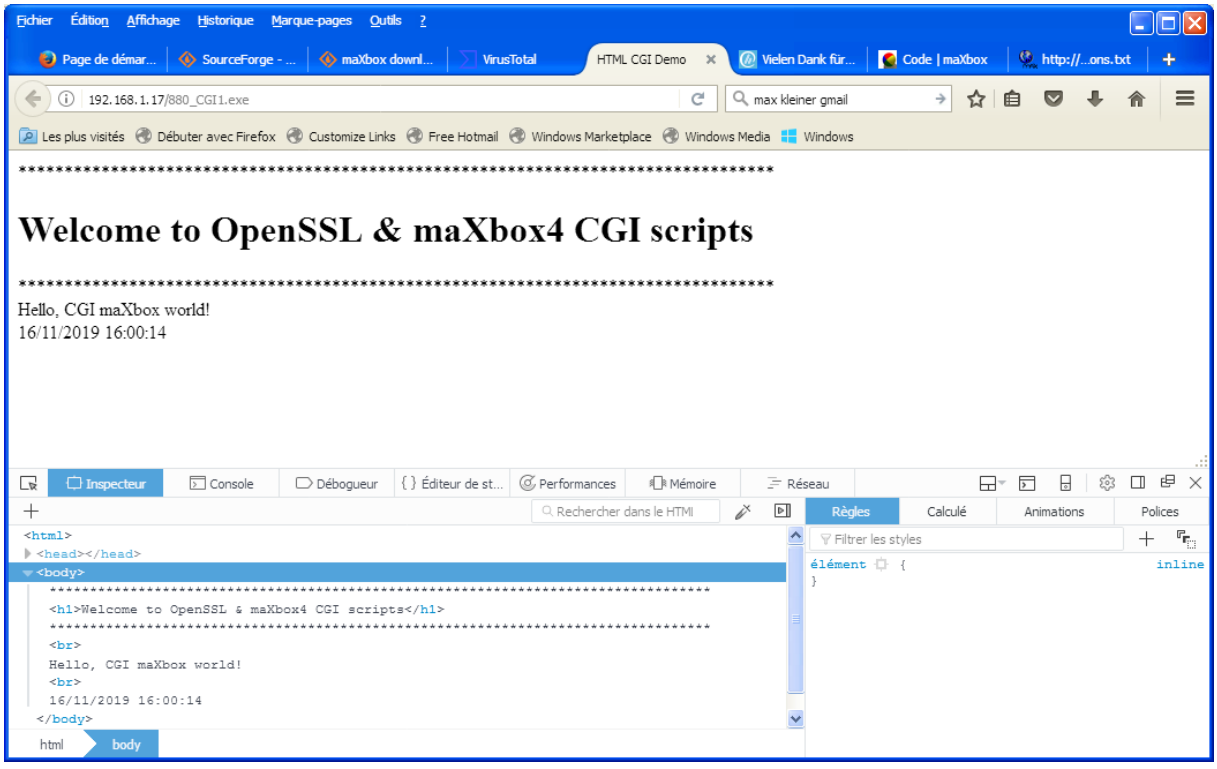
we must do something special. In fact, there are three common ways to safe state information: FAT URLs, hidden fields and cookies.
We just start our compiled CGI with the following URL:

http://192.168.1.66/cgi-bin/CGI1.exe

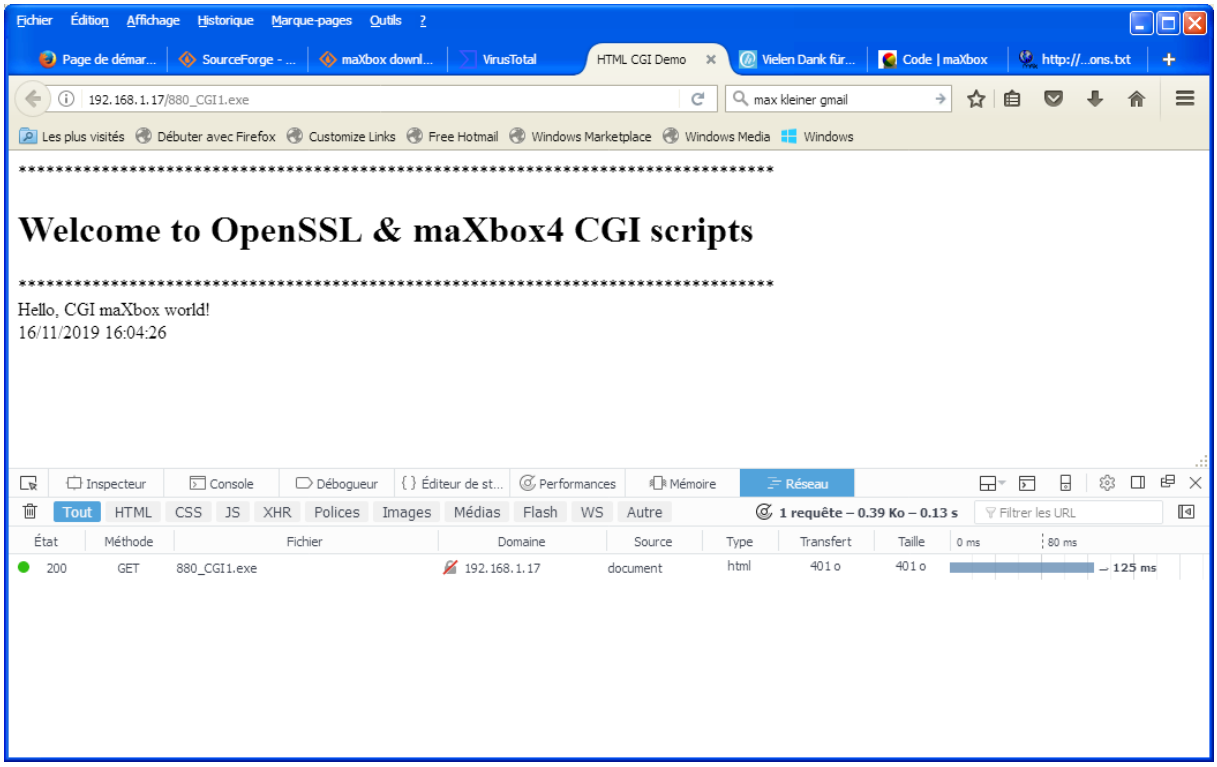
The code object is an Indy Extension called `TidCGIRunner`:

```
//object CGIRunner: TidCGIRunner
  CGIRunner:= TidCGIRunner.create(self)
    with CGIRunner do begin
      Server := HTTPServer
      TimeOutMsg := '<html><body><h1><center>Process is
terminated.</body></html>'#13#10
      ErrorMsg := '<html><body><h1><center>Internal Server
Error</body></html>'#13#10
      ServerAdmin := 'admin@server'
      PHPSupport:= true;
      //RegisterProperty('PHPIniPath', 'String', iptrw);
      //Left := 260
      //Top := 125
  end //}
```

*Another benefit to CGI is that it is web server agnostic. The same **executable** will work for IIS, nginx as it does for Apache, as it does for any other W3C compliant web server. And unlike some scripting languages, the executable is compiled so execution is fast and the source is not visible. So calling the CGI-extension with F12 Inspection looks like:*



One reason you might note is that a CGI application is executed as a separate process every time a web page requests it. This is unlike some forms of Apache modules or ISAPI IIS DLLs where the CGI application is loaded once and stays in memory.




Another reason you may avoid using CGI is that the burden for security is higher with CGI than it is with a server-side scripting language. There are limitations in scripting languages that are not present in your own application, but Pascal or PHP are rich of function features.
Because of that, extra caution needs to be applied, especially in how you handle input coming from (you hope) your users. Check out the section on security to learn about what methods are used to make your CGI app tight.

https://www.codeproject.com/articles/9433/understanding-cgi-with-c

```
//object HTTPServer: TIdHTTPServer
    HTTPServer:= TIdHTTPServer.create(self)
    with httpserver do begin
     HTTPServer.bindings.Clear;
     HTTPServer.OnStatus:= @TfmHTTPServerMainHTTPServerStatus;
     defaultport:= PORT_Socket; //8090;
     //Bindings.Add;
      bindings.add.port:= defaultport;
      bindings.add.ip:= IP_Socket;
```

So, we create a own CGI runner and It works well, if I set the working directory to child process created for my CGI runner executing a compiled server side scripting **CGI1.exe**.

You can find the script-code at:

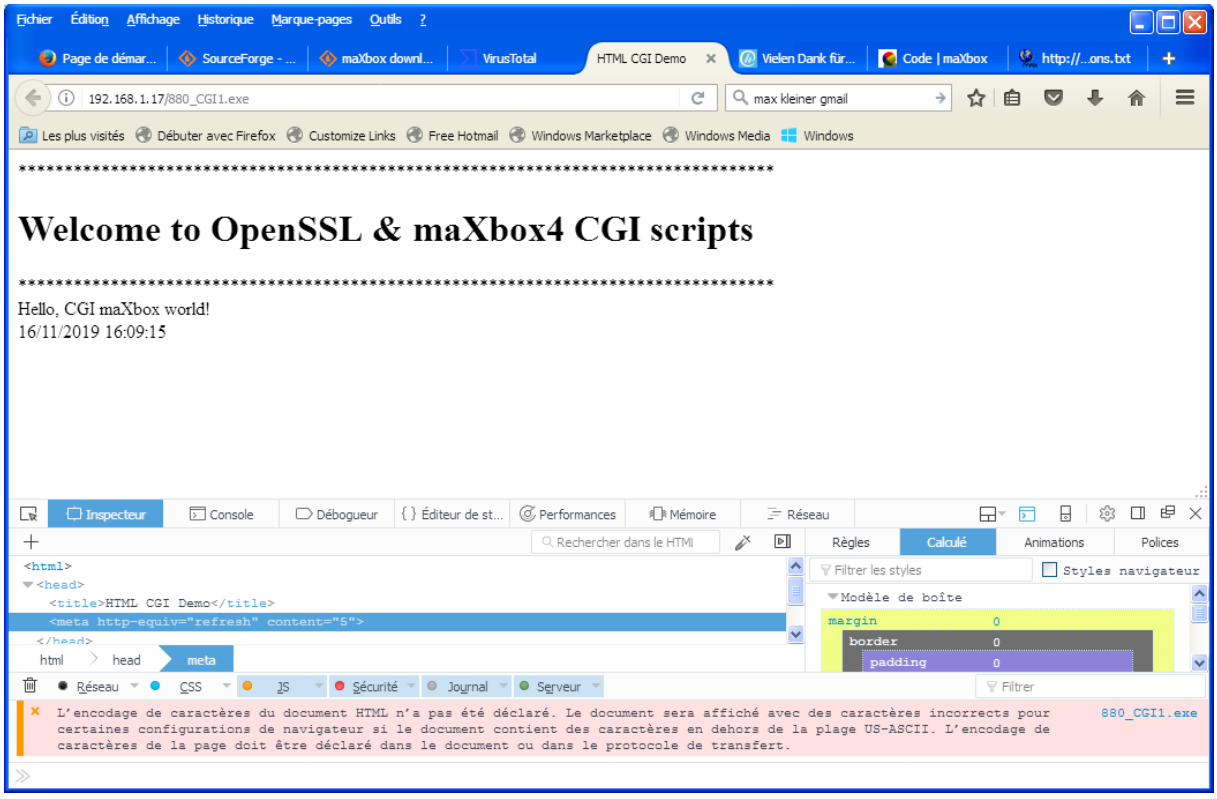
http://www.softwareschule.ch/examples/cgi.txt

The whole project with the source and compiled exe at:

https://sourceforge.net/projects/maxbox/files/Examples/13_General/880_cgiMain_server3.pas/download

So you have seen CGI in all sorts of places on the Web, although you may not have known it at the time. For example:

- Any guest book allows you to enter a message in an HTML form and then, the next time the guest book is viewed, the page will contain your new entry.
- The WHOIS form at Network Solutions allows you to enter a domain name on a form, and the page returned is different depending on the domain name entered.
- Any search engine lets you enter keywords on an HTML form, and then it dynamically creates a page based on the keywords you enter.




https://www.bytesin.com/software/maXbox/

## 1.2 Web Sockets

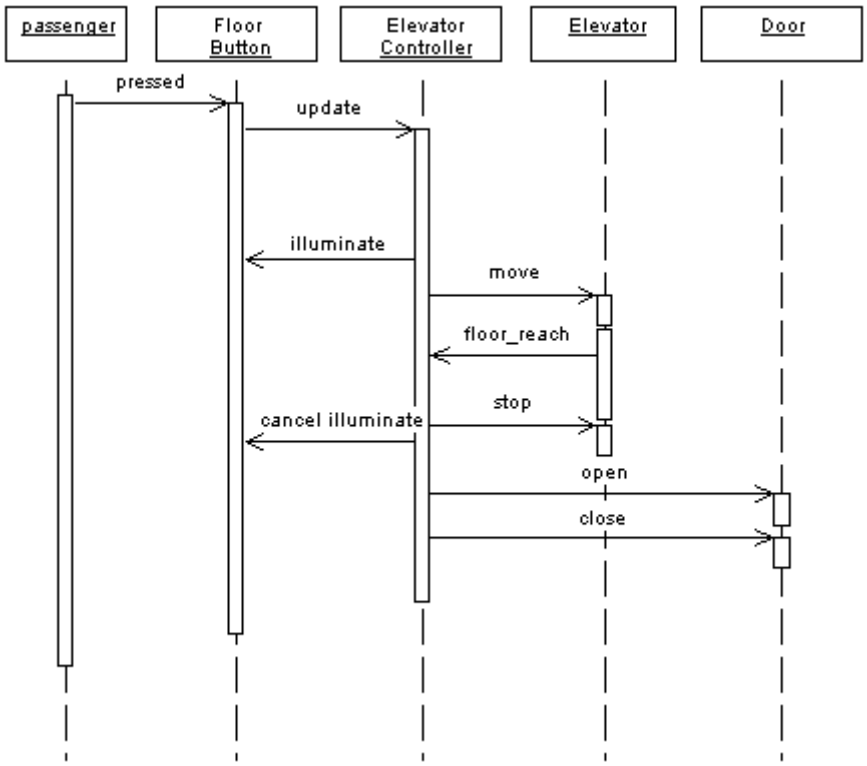
- According to the draft specification, the Web Socket protocol enables two-way communication between a user agent running untrusted code running in a controlled environment to a remote host that has opted-in to communications from that code. The security model used for this is the Origin-based security model commonly used by Web browsers. The protocol consists of an initial handshake followed by basic message framing, layered over TCP. The goal of this technology is to provide a mechanism for browser-based applications that need two-way communication with servers that does not rely on opening multiple HTTP connections (e.g. using XMLHttpRequest or iframes and long polling).

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

```
// The TWebSocketConnection represents the communication
 // chanel with a single connected client. When the connection
 // is innitiated, the handshake procedure is executed. If the
 // handshake succedes the channel can be used for sending and
 // receiving. If not, the connection must be closed.

 TWebSocketConnection = class
 private
   FPeerThread: TIdPeerThread;
   FHandshakeRequest: TWebSocketRequest;
   FHandshakeResponseSent: Boolean;
   FOnMessageReceived: TWebSocketMessageEvent;
   function GetHandshakeCompleted: Boolean;
   function GetServerConnection: TIdTCPServerConnection;
   function GetPeerIP: string;
 protected
   const
     FRAME_START = #$00;
     FRAME_SIZE_START = #$80;
     FRAME_END = #$FF;

   procedure Handshake;
   procedure SendFrame(const AData: string);

   property PeerThread: TIdPeerThread read FPeerThread write FPeerThread;
   property ServerConnection: TIdTCPServerConnection read
                                                 GetServerConnection;
   property HandshakeRequest: TWebSocketRequest read FhandshakeRequest
                                        write FHandshakeRequest;
```

```
    property HandshakeCompleted: Boolean read GetHandshakeCompleted;
    property HandshakeResponseSent: Boolean read FhandshakeResponseSent
                                    write FHandshakeResponseSent;
  public
    constructor Create(APeerThread: TIdPeerThread);

    procedure Receive;
    procedure Send(const AMessage: string);

    property OnMessageReceived: TWebSocketMessageEvent read
                               FOnMessageReceived write FOnMessageReceived;
    property PeerIP: string read GetPeerIP;
  end;
```

For example in the following sequence-diagram a passenger is the browser with a floor button. The elevator is the server and after handshaking a communication the server pushes commands like illuminate to the floor.




A WebSocket server is an application listening on any port of a TCP server that follows a specific protocol, simple as that. The task of creating a custom server tends to scare people; however, it can be easy to implement a simple WebSocket server on your platform of choice.

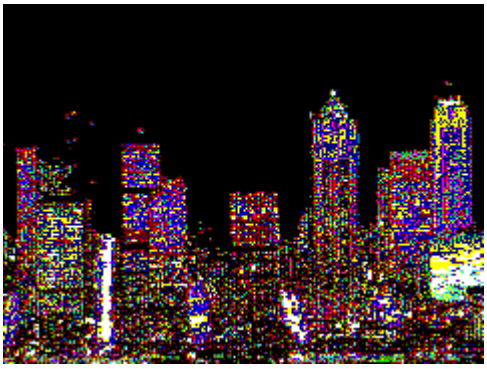
```
// The TWebSocketServer is a TCP server "decorated" with
  // some websocket specific functionalities.
  TWebSocketServer = class
  private
    FDefaultPort: Integer;
```

```
FTCPServer: TIdTCPServer;
FThreadManager: TIdThreadMgr;
FConnections: TObjectList;
FOnConnect: TWebSocketConnectEvent;
FOnMessageReceived: TWebSocketMessageEvent;
FOnDisconnect: TWebSocketDisconnectEvent;
function GetTCPServer: TIdTCPServer;
function GetThreadManager: TIdThreadMgr;
function GetConnections: TObjectList;
function GetActive: Boolean;
procedure SetActive(const Value: Boolean);
```

Code and script can be found at:
http://www.softwareschule.ch/examples/751_Elevator_Simulator4.pas

http://www.softwareschule.ch/examples/766_wmi_management.txt

Conclusion:
The handshake is the "Web" in WebSockets. It's the bridge from HTTP to WebSockets. In the handshake, details of the connection are negotiated, and either party can back out before completion if the terms are un-favorable. The server must be careful to understand everything the client asks for, otherwise security issues will be introduced.



"Wise speak: Better late than wait.

Feedback @ max@kleiner.com
Literature: Kleiner et al., Patterns konkret, 2003, Software & Support

https://github.com/maxkleiner/maXbox4/releases

https://www.academia.edu/31112544/Work_with_microservice_maXbox_starter48.pdf

Binary of CGI Exe:
https://sourceforge.net/projects/maxbox/files/Examples/13_General/880_CGI1.exe/download

## 1.3 References

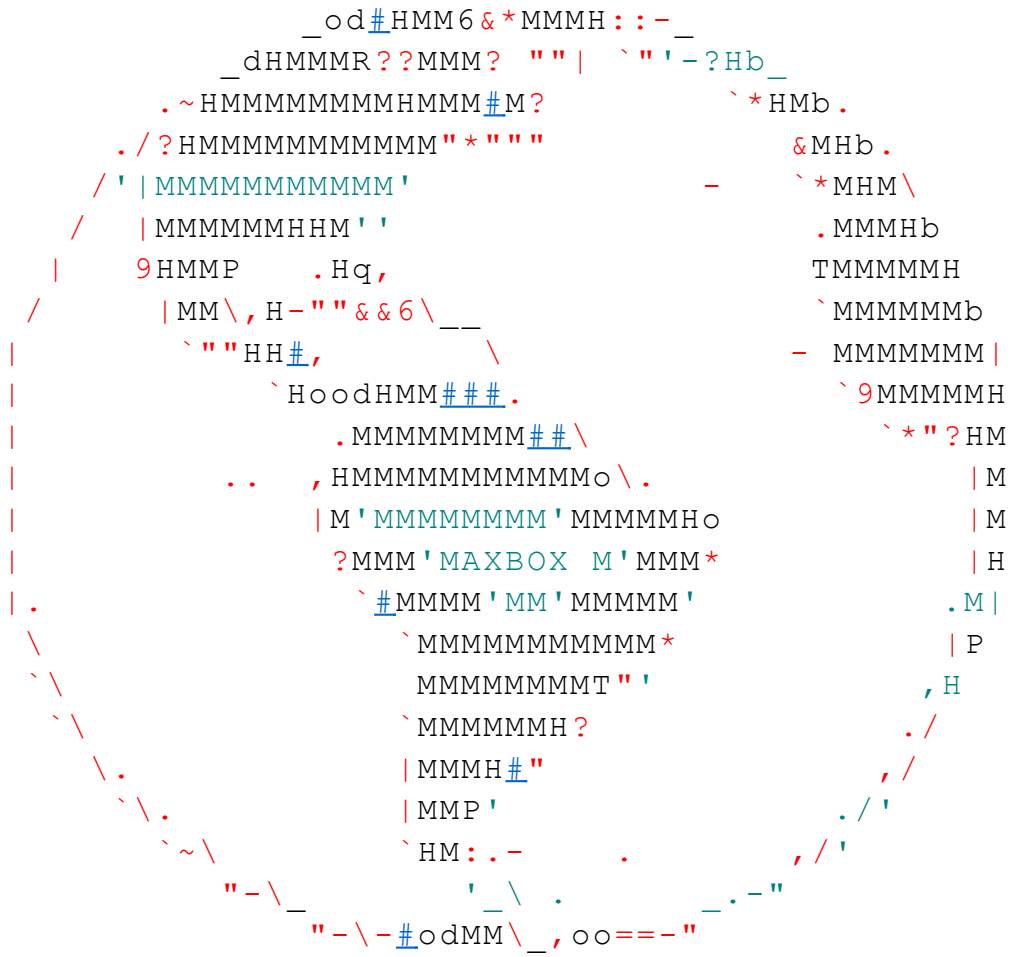
Examples and Version of this Tutorial 71:
2017-07 Cumulative Update for Windows 10 Version 1703 for x64-based Systems (KB4025342) – maXbox 4.7.1.20

https://en.wikipedia.org/wiki/WebSocket

http://www.softwareschule.ch/examples/cgi.txt

www.softwareschule.ch/examples/751_Elevator_Simulator3.pas

http://www.softwareschule.ch/examples/766_wmi_management.txt



Talk to each other across the globe, in a human voice makes the race.

https://maxbox4.wordpress.com/