ESP32Forth simple WEB interface.

Note - next info based on use of version 7.0.7.5, tested also on 7.0.7.15.
I restarted Forth usage after years out of any programming with FlashFORTH on Atmega 328 and Arduino. After creating my first construction it was necessary to build some control panel for electronics, some buttons, display aso. I have thought there is time of IoT and wireless control, so better spare construction work and control all wireless. For this I moved to ESP32 with WiFi and BT, I have found tens of program examples of web interfaces in Arduino C with JavaScript, but nothing in ESP32Forth on ESP32. For me as beginner it was problem.
So next is result of my effort - simple example of web interface, on web server running on ESP32 in Forth. Code is based on Peter Forth example *peter-webpage-dht11-graphic-example.txt.*
Whole code is in attached file *example_web.fs.,* line numbers are from this file.
Web server runs on ESP32 board with activated WiFi connection and responds to client (browser on PC, mobile etc.) requests.
So basic web interface is simple:

```
178
179 : runpage  begin handleClient if serve-page 100 ms then 500 ms again ;
```

where *handleClient* detects if there are client requests, resolves request and gives HTML content to client with word *serve-page*. This *ms* delays improved wifi connection stability in my home net.

```
164
165 : serve-page  ( --)              \ simple parsing and action of client respond
166   path s" /" str= if htmlpagesend exit then \ exit leaves from serve-page
167   path s" /26/on" str= if cr ." ACTION for  /26/on " cr \ here put action word
168    0 to GPIO26 htmlpagesend exit then
169   path s" /26/off" str= if cr ." ACTION for /26/off " cr
170    1 to GPIO26 htmlpagesend exit then
171     path s" /27/on" str= if cr ." ACTION for /27/on " cr
172    0 to GPIO27 htmlpagesend exit then
173   path s" /27/off" str= if cr ." ACTION for /27/off " cr
174    1 to GPIO27 htmlpagesend exit then
175   path respond      \ actions for html forms
176   htmlpagesend exit  \ resend html page
177 ;
```

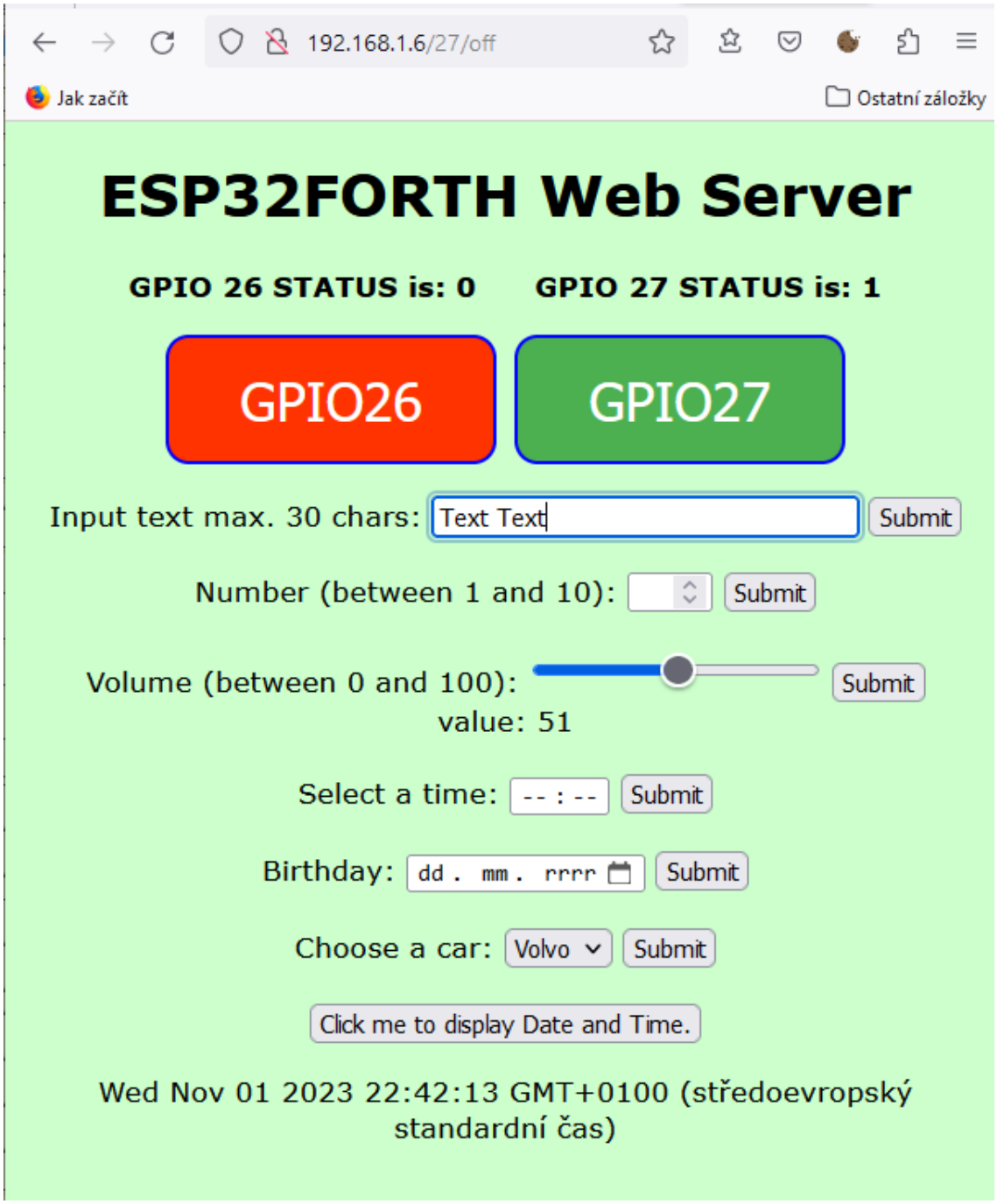
Serve-page uses text of client request from word *path* in form *addr len* and compares it with possible client responds, each match activates relevant action and refreshes HTML page content with word *htmlpagesend.* Action word(s) can be put instead of substitutes as is *." ACTION for /26/on " aso.*

```
136
137 : htmlpagesend  \ send whole html page
138     s" text/html" ok-response
139     htmlpage    \ create html page in webintstream buffer
140     webcontent send    \ and send it to client
141 ;
```

Htmlpagesend sends back to client (browser) at first status code and type of html data. Next is dynamically created html page code in form of text and finally sent to client to show it in browser. This is whole process in the nutshell.
Next in more detail.
For practical usage I focused to 3 types of information generated by ESP32 web interface:
- simple passive text data such as results of some measuring, for example from weather station

- buttons for on/off switching to control something by ESP32 circuit
- HTML forms for adjusting some parameters in program running on ESP32.

For this I created this simple example of web page generated on ESP32 with IP address 192.168.1.6.



So text **GPIO 26 status is:** is text info and value **0 or 1** is value of forth value *GPIO26* included to web page during HTML page generation.
Buttons GPIO26 and GPIO27 can switch appropriate forth value to control something, for ex. relay connected to ESP32 GPIOs.
The rest HTML forms can control more advanced adjusting of forth program parameters.

The last Click me to display… is only generating actual date/time info of client browser without any program connection to ESP32forth code.
Next only in brief:
Lines 8 to 29 create helping word mvbar, used as ***mvbar*** *any multi line text* **|** to create temporary string as *addr len* across more lines of text.
Buffer for HTML page text is created on line 31 by stream *webintstream* and uses word >*stream* to add text parts together.
Long word *htmlpage* on lines 46 to 135 dynamically creates html text after each activation. Lines 67 to 71 create text with actual values of forth *values GPIO26, GPIO27*. If it is used to show some measured values continually it is necessary to put code for auto refresh of html page into generated html code.
Lines 72 to 88 create buttons in color red or green depending on GPIO values with client info /26/on or /26/off for detection in *serve-page* word.
Lines 91 to 127 generate html forms data for adjustment of some values as data, time, text or range.
Lines 128 to 131 are only generating actual date/time info with JavaScript code.
In the end of code there is activation of server with wifi and start of webinterface as task on background.
I present this code as base for experiments. I am sure it is possible to improve it, comments are welcome.

example-web.fs

```
\ Language: ESP32FORTH
\ Program for simple web interface
\ Author: Vaclav Poselt October 2023
defined? MARKER 0<> [if] forget MARKER [then]
create MARKER
only also httpd also streams also internals

\ next is definition of helping word mvbar
0 value mystr       \ addres of string block
0 value mystr#      \ string block position?
0 value mystrn      \ string block lenght
: mystr-init ( -- ) \ activate 10 bytes buffer
   10 dup allocate ( 10 addr ior )
   throw ( 10 addr )
   to mystr to mystrn 0 to mystr# ; \ addr to mystr, 10 to mystrn, 0 to mystr#
: mystr-grow ( -- ) \ increase buffer to (size+1)*2
   mystrn 1+ 2* to mystrn mystr mystrn resize throw to mystr ;
: +mystr ( ch -- ) \ add char to buffer
   mystr# mystrn >= if mystr-grow then  \ increase buffer if necessary
   mystr mystr# + c! 1 +to mystr# ;     \ put char to next position in buffer
: 1tch   begin >in @ #tib @ >= while refill drop nl +mystr repeat ;
: tch ( -- ch ) tib >in @ + c@ ;
: vbar? ( -- f ) 1tch tch 124 = ;  \ 124 is code for vertical bar symbol
: m$ mystr-init
    begin vbar? 0= while tch +mystr 1 >in +! repeat 1 >in +!
    mystr mystr# ;
: mvbar ( comp: -- <string|> | exec: addr len) \ create temporary counted string
   m$ state @ if swap [ internals ] aliteral aliteral then ; immediate
```

```
\ end of definition of mvbar |

3000 stream webintstream                \ buffer for user web interface content
\ where 1st cell is total lenght of buffer, 2nd cell no of written bytes, 4rth
\ cell is start of buffer content
: webcontent    ( -- addr len )        \ prepare values for html send
   webintstream dup 3 cells +          \ gives addr of first byte
   swap cell+ @                        \ gives no of written bytes
;
: webcontentreset ( -- )               \ reset stream
   webintstream  cell+ 0 swap !        \ reset no of written bytes
;

0 value GPIO26  \ staus of GPIO26, 0-ON, 1-off
0 value GPIO27  \ staus of GPIO27, 0-ON, 1-off
\ pins GPIO26. 27 can be used for control something

: htmlpage      \ create whole html page in webintstream
webcontentreset \ reset html page buffer
mvbar <!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="icon" href="data:,">
<!-- CSS to style the on/off buttons -->
<!-- Feel free to change the background-color and font-size attributes to fit your preferences -->
<style>
html { font-family: Verdana, Helvetica; display: inline-block; margin: 0px auto; text-align: center;
background-color: #CCFFCC}
.button { background-color: #4CAF50; border: 2px solid blue; border-radius: 12px;
color: white; padding: 16px 40px; text-decoration: none; font-size: 30px; margin: 2px;
cursor: pointer;}
.button2 {background-color: #FF3300;}
</style>
</head>
<body><h1>ESP32FORTH Web Server</h1> | webintstream >stream  \ common part of HTML
page

\ Display current state, and ON/OFF buttons for GPIO 26 and 27
mvbar <p><b>GPIO 26 STATUS is: | webintstream >stream
GPIO26 str webintstream >stream    \ show value of GPIO26
mvbar       GPIO 27 STATUS is: | webintstream >stream
GPIO27 str webintstream >stream    \ show value of GPIO27
mvbar </b></p> | webintstream >stream
\ display buttons in color acording  value
GPIO26 1 = if   \ if OFF show ON button else OFF button
        mvbar <p><a href="/26/on"><button class="button">GPIO26</button></a>
             | webintstream >stream
        else
         mvbar <p><a href="/26/off"><button class="button button2">GPIO26</button></a>
```

```
          | webintstream >stream
         then


GPIO27 1 = if   \ if OFF show ON button else OFF button
        mvbar <a href="/27/on"><button class="button">GPIO27</button></a></p>
             | webintstream >stream
        else
         mvbar <a href="/27/off"><button class="button button2">GPIO27</button></a></p>
          | webintstream >stream
         then

mvbar
<form action="/get">
  <label for="txt">Input text max. 30 chars: </label>
  <input type="text" id="txt" name="TX" size="30">
  <input type="submit" value="Submit">
</form><br>
<form action="/get">
  <label for="num">Number (between 1 and 10): </label>
  <input type="number" id="num" name="NO" min="1" max="10" size="4">
  <input type="submit" value="Submit">
</form><br>
<form action="/get">
  <label for="vol">Volume (between 0 and 100): </label>
  <input type="range" name="RG" value="4" min="0" max="100" id="vol"
  onchange="document.getElementById('ran').innerText = this.value" >
  <input type="submit" value="Submit">
  <br> value: <span id="ran"></span>
</form><br>
<form action="/get">
  <label for="appt">Select a time:</label>
  <input type="time" id="appt" name="TM">
  <input type="submit" value="Submit">
</form><br>
 <form action="/get">
  <label for="dat">Birthday:</label>
  <input type="date" id="dat" name="DT">
  <input type="submit" value="Submit">
</form><br>
<form  action="/get">
  <label for="cars">Choose a car:</label>
  <select id="cars" name="CA">
    <option value="volvo">Volvo</option>
    <option value="saab">Saab</option>
    <option value="fiat">Fiat</option>
    <option value="audi">Audi</option>
  </select>
  <input type="submit" value="Submit">
</form><br>
```

```
<button type="button"
onclick="document.getElementById('demo').innerHTML = Date()">
Click me to display Date and Time.</button><br>
<p id="demo"></p>
</body>
</html>
  | webintstream >stream     \ last part of html page saved to webintstream
;

: htmlpagesend  \ send whole html page
   s" text/html" ok-response
   htmlpage    \ create html page in webintstream buffer
   webcontent send    \ and send it to client
;
create mypad 8 allot          \ create 8 bytes buffer
\ respond  will analyze given path and decode returned values from html forms
\ with possible code for actions, now there is only print of values
: respond  ( addr len-- )
   dup 50 min 8 >             \ len is min 9 chars or more, for. ex. /get?IN=x
   if over     ( addr len addr --)
     mypad  8 cmove      \ first 8 chars to mypad
     swap 8 + swap 8 - ( addr+8 len-8--) \ from 9th char of path is returned value
     s" /get?TX=" mypad 8 str=    \ it is text value
       if cr ." text value:"  type then
     s" /get?NO=" mypad 8 str=    \ it is number
       if cr ." number value:"  s>number? drop . then
     s" /get?RG=" mypad 8 str=    \ it is range value
       if cr ." range value:"  s>number? drop . then
     s" /get?TM=" mypad 8 str=    \ it is time value as 22%3A39=22:39
       if cr ." time value:" drop dup 2 type [char] : emit 5 + 2 type then
     s" /get?DT=" mypad 8 str=    \ it is date value
       if cr ." date value:"  type then
     s" /get?CA=" mypad 8 str=    \ it is select value
       if cr ." select value:"  type then
     then
;

: serve-page  ( --)             \ simple parsing and action of client respond
  path s" /" str= if htmlpagesend exit then \ exit leaves from serve-page
  path s" /26/on" str= if cr ." ACTION for  /26/on " cr \ here put action word
   0 to GPIO26 htmlpagesend exit then
  path s" /26/off" str= if cr ." ACTION for /26/off " cr
   1 to GPIO26 htmlpagesend exit then
    path s" /27/on" str= if cr ." ACTION for /27/on " cr
   0 to GPIO27 htmlpagesend exit then
  path s" /27/off" str= if cr ." ACTION for /27/off " cr
   1 to GPIO27 htmlpagesend exit then
  path respond      \ actions for html forms
  htmlpagesend exit  \ resend html page
;
```

```
: runpage  begin handleClient if serve-page 100 ms then 500 ms again ;
: connect z" your SSID" z" your psw" login 80 server ;

' runpage 100 100 task webtask
: runServer ( -- ) connect webtask start-task ; \
```