```
1
2   // https://github.com/me-no-dev/ESPAsyncTCP
3   // https://github.com/me-no-dev/AsyncTCP
4   // https://github.com/adafruit/Adafruit_NeoMatrix
5   // https://github.com/adafruit/Adafruit_NeoPixel
6
7
8    // SPIFF Opload with PlatformIO
9    // pio run -t uploadfs
10
11  #include <ArduinoOTA.h>
12  #ifdef ESP32
13  #include <FS.h>
14  #include <SPIFFS.h>
15  #include <ESPmDNS.h>
16  #include <WiFi.h>
17  #include <AsyncTCP.h>
18  #elif defined(ESP8266)
19  #include <ESP8266WiFi.h>
20  #include <ESPAsyncTCP.h>
21  #include <ESP8266mDNS.h>
22  #endif
23  #include <ESPAsyncWebServer.h>
24  #include <SPIFFSEditor.h>
25
26
27
28  // Adafruit_NeoMatrix example for tiled NeoPixel matrices.  Scrolls
29  // 'Howdy' across three 10x8 NeoPixel grids that were created using
30  // NeoPixel 60 LEDs per meter flex strip.
31
32  #include <Adafruit_GFX.h>
33  #include <Adafruit_NeoMatrix.h>
34  #include <Adafruit_NeoPixel.h>
35  #include "ILI9341_Colors.h"
36
37
38  #include "WiFiconig.inc"
39
40  // SKETCH BEGIN
41  AsyncWebServer server(80);
42  AsyncWebSocket ws("/neopixel");
43  AsyncEventSource events("/events");
44
45
46  // MATRIX DECLARATION:
47  // Parameter 1 = width of EACH NEOPIXEL MATRIX (not total display)
48  // Parameter 2 = height of each matrix
49  // Parameter 3 = number of matrices arranged horizontally
50  // Parameter 4 = number of matrices arranged vertically
51  // Parameter 5 = pin number (most are valid)
52  // Parameter 6 = matrix layout flags, add together as needed:
53  //   NEO_MATRIX_TOP, NEO_MATRIX_BOTTOM, NEO_MATRIX_LEFT, NEO_MATRIX_RIGHT:
54  //     Position of the FIRST LED in the FIRST MATRIX; pick two, e.g.
55  //     NEO_MATRIX_TOP + NEO_MATRIX_LEFT for the top-left corner.
56  //   NEO_MATRIX_ROWS, NEO_MATRIX_COLUMNS: LEDs WITHIN EACH MATRIX are
57  //     arranged in horizontal rows or in vertical columns, respectively;
58  //     pick one or the other.
59  //   NEO_MATRIX_PROGRESSIVE, NEO_MATRIX_ZIGZAG: all rows/columns WITHIN
60  //     EACH MATRIX proceed in the same order, or alternate lines reverse
61  //     direction; pick one.
62  //   NEO_TILE_TOP, NEO_TILE_BOTTOM, NEO_TILE_LEFT, NEO_TILE_RIGHT:
63  //     Position of the FIRST MATRIX (tile) in the OVERALL DISPLAY; pick
64  //     two, e.g. NEO_TILE_TOP + NEO_TILE_LEFT for the top-left corner.
65  //   NEO_TILE_ROWS, NEO_TILE_COLUMNS: the matrices in the OVERALL DISPLAY
66  //     are arranged in horizontal rows or in vertical columns, respectively;
67  //     pick one or the other.
68  //   NEO_TILE_PROGRESSIVE, NEO_TILE_ZIGZAG: the ROWS/COLUMS OF MATRICES
69  //     (tiles) in the OVERALL DISPLAY proceed in the same order for every
70  //     line, or alternate lines reverse direction; pick one.  When using
71  //     zig-zag order, the orientation of the matrices in alternate rows
72  //     will be rotated 180 degrees (this is normal -- simplifies wiring).
73  //   See example below for these values in action.
```

```
74  // Parameter 7 = pixel type flags, add together as needed:
75  //   NEO_RGB     Pixels are wired for RGB bitstream (v1 pixels)
76  //   NEO_GRB     Pixels are wired for GRB bitstream (v2 pixels)
77  //   NEO_KHZ400  400 KHz bitstream (e.g. FLORA v1 pixels)
78  //   NEO_KHZ800  800 KHz bitstream (e.g. High Density LED strip)
79
80  // Example with three 10x8 matrices (created using NeoPixel flex strip --
81  // these grids are not a ready-made product).  In this application we'd
82  // like to arrange the three matrices side-by-side in a wide display.
83  // The first matrix (tile) will be at the left, and the first pixel within
84  // that matrix is at the top left.  The matrices use zig-zag line ordering.
85  // There's only one row here, so it doesn't matter if we declare it in row
86  // or column order.  The matrices use 800 KHz (v2) pixels that expect GRB
87  // color data.
88
89  Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(  NeoPixelTileWidth,
    NeoPixelTileHeight, NeoTilesModulesX, NeoTilesModulesY, NeoPixelPIN,
90
91  //Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(NeoPixelTileWidth,
    NeoPixelTileHeight, 3, 1, NeoPixelPIN,
92
93    NEO_MATRIX_BOTTOM    + NEO_MATRIX_LEFT +
94    NEO_MATRIX_ROWS + NEO_MATRIX_PROGRESSIVE +
95
96    NEO_TILE_TOP + NEO_TILE_LEFT + NEO_TILE_COLUMNS + NEO_TILE_PROGRESSIVE ,
97
98
99    NEO_GRBW + NEO_KHZ800);
100
101 const uint16_t colors[] = {
102   matrix.Color(255, 0, 0), matrix.Color(0, 255, 0), matrix.Color(0, 0, 255)
103   };
104
105 uint16_t RGB32_2RGB16(String Value)
106 {
107   Value.replace("hex","");
108     uint32_t rgb32 = strtoul(Value.c_str(), 0, 16);
109   //Serial.printf("-----> %s <----$$$####> %d <-----", Value.c_str(), rgb32 );
110     return (rgb32 >> 8 & 0xf800) | (rgb32 >> 5 & 0x07e0) | (rgb32 >> 3 & 0x001f);
111 }
112
113 void InitPorts();
114 void SetupNeoPixel();
115 void SetupOTA();
116 void onWsEvent(AsyncWebSocket * server, AsyncWebSocketClient * client, AwsEventType
    type, void * arg, uint8_t *data, size_t len);
117 void connectToWiFi(const char * ssid, const char * pwd);
118 void SetupServer();
119 String HTML_Processor(const String& var);
120
121
122 //// NeoPixel Pattern
123 void NeoPixelTest();
124 void NeoPixelPattern1();
125 void NeoPixelPattern2();
126 void NeoPixelPattern3();
127 void NeoPixelPattern4();
128 void NeoPixelPattern5();
129 void NeoPixelPattern6();
130 void NeoPixelPattern7();
131 void NeoPixelPattern8();
132 void NeoPixelPattern9();
133 void NeoPixelPattern10();
134 void NeoPixelPattern11();
135 void NeoPixelPattern12();
136
137 void CheckPattern(MatrixMessage_s Value);
138 void NeoPixelText1(const MatrixMessage_s Value);
139 MatrixMessage_s GetMatrixTextValues(String Value);
140
141
142 /*************************************************************************************
143 * @brief  setup()
```

```
144   * @return none
145   ****************************************************************************/
146   void setup() {
147      Serial.begin(115200);
148      Serial.setDebugOutput(true);
149      ESP.wdtDisable();
150      InitPorts();
151
152       connectToWiFi(ssid, password);
153       SetupServer();
154       SetupOTA();
155       SetupNeoPixel();
156       NeoPixelTest();
157
158       ESP.wdtEnable(1000);
159   }
160
161
162
163   void loop() {
164     ESP.wdtFeed();
165     if (ColorRGB16 != MerkerColorRGB16)
166     {
167     matrix.fillRect(0,0, NeoPixelMatrixWidth, NeoPixelTileHeight,ColorRGB16);
168     matrix.show();
169     delay(50);
170     MerkerColorRGB16 = ColorRGB16;
171
172     }
173      ArduinoOTA.handle();
174    if (sWebSocket.bTriggerWebsocketMsg)
175    {
176   sWebSocket.bTriggerWebsocketMsg = false;
177   sprintf(Buffer, "[Status] %s", sWebSocket.WebsocketMsg.c_str()  );
178
179    Serial.printf ("[in Loop ---- > %s -Trigger: %d--Color : 0x%x -- Bkg 0x%x ---
       Pattern : %d --- \n", sMatrixMessage.Text.c_str(), sMatrixMessage.bTriggerPattern,
       sMatrixMessage.Color, sMatrixMessage.BkgColor, sMatrixMessage.Pattern);
180   ws.printf(ClientID, Buffer );
181   events.send("In the Loop",NULL,millis(),500);
182    }
183
184   CheckPattern(sMatrixMessage);
185      ws.cleanupClients();
186
187      if (ToggleCounter++ > 10000)
188   {
189      ToggleCounter = 0;
190      TOGGLE(LedStatus1);
191   }
192   }
193
194
195
196   void InitPorts()
197   {
198
199   pinMode( LedStatus1, OUTPUT);
200   pinMode( LedStatus2, OUTPUT);
201
202   }
203
204
205
206   /****************************************************************************
207   * @brief  CheckPattern
208   * @return none
209   ****************************************************************************/
210   void CheckPattern(MatrixMessage_s Value)
211   {
212
213   if ( Value.bTriggerPattern)
214   {
```

```
215     TOGGLE(LedStatus2);
216     Serial.printf ("[---- > %s -Trigger: %d--Color : 0x%x -- Bkg 0x%x --- Pattern : %d
        --- \n", Value.Text.c_str(), Value.bTriggerPattern, Value.Color, Value.BkgColor,
        Value.Pattern);
217   ws.printf(ClientID, "[BuDisable]" );
218
219   switch (Value.Pattern)
220   {
221   case 1: NeoPixelPattern1();    break;
222   case 2: NeoPixelPattern2();    break;
223   case 3: NeoPixelPattern3();    break;
224   case 4: NeoPixelPattern4();    break;
225   case 5: NeoPixelPattern5();    break;
226   case 6: NeoPixelPattern6();    break;
227   case 7: NeoPixelPattern7();    break;
228   case 8: NeoPixelPattern8();    break;
229   case 9: NeoPixelPattern9();    break;
230   case 10: NeoPixelPattern10();    break;
231   case 11: NeoPixelPattern11();    break;
232   case 12: NeoPixelPattern12();    break;
233   case 13: NeoPixelText1(Value);    break;
234
235   default:
236     break;
237   }
238
239   matrix.fillScreen(ILI9341_BLACK);
240   matrix.show();
241
242   sMatrixMessage.bTriggerPattern = false; // We can change this later
243   ws.printf(ClientID, "[BuEnable]" );
244   }
245   }
246
247   /**************************************************************************
248   * @brief  SetupNeoPixel()
249   * @return none
250   **************************************************************************/
251   void SetupNeoPixel()
252   {
253     matrix.begin();
254     matrix.setTextWrap(false);
255     matrix.setBrightness(20);
256     matrix.setTextColor(colors[0]);
257   }
258
259
260   /**************************************************************************
261   * @brief  connectToWiFi
262   * @return none
263   **************************************************************************/
264   void connectToWiFi(const char * ssid, const char * pwd)
265   {
266           Serial.printf(Buffer, "Connecting to WiFi network : %s", ssid);
267
268     // delete old config
269     WiFi.disconnect(true);
270      WiFi.hostname(hostName);
271
272    if (!WiFi.config(staticIP, gateway, subnet, dns1, dns2))
273    {
274    Serial.println("STA Failed to configure");
275    }
276
277     //Initiate connection
278     WiFi.begin(ssid, pwd);
279
280     while (WiFi.status() != WL_CONNECTED)
281     {
282       delay(150);
283       Serial.print(".");
284     }
285
```

```
286     sIPAddress =  WiFi.localIP().toString();
287
288     Serial.printf("************************IP Adresse %s ********************\n",
        sIPAddress.c_str());
289       }
290
291
292   /**************************************************************************************
293   * @brief  SetupOTA()
294   * @return none
295   **************************************************************************************/
296   void SetupOTA()
297   {
298     //Send OTA events to the browser
299     ArduinoOTA.onStart([]() { events.send("Update Start", "ota"); });
300     ArduinoOTA.onEnd([]() { events.send("Update End", "ota"); });
301     ArduinoOTA.onProgress([](unsigned int progress, unsigned int total) {
302       char p[32];
303       sprintf(p, "Progress: %u%%\n", (progress/(total/100)));
304       events.send(p, "ota");
305     });
306     ArduinoOTA.onError([](ota_error_t error) {
307       if(error == OTA_AUTH_ERROR) events.send("Auth Failed", "ota");
308       else if(error == OTA_BEGIN_ERROR) events.send("Begin Failed", "ota");
309       else if(error == OTA_CONNECT_ERROR) events.send("Connect Failed", "ota");
310       else if(error == OTA_RECEIVE_ERROR) events.send("Recieve Failed", "ota");
311       else if(error == OTA_END_ERROR) events.send("End Failed", "ota");
312     });
313     ArduinoOTA.setHostname(hostName);
314     ArduinoOTA.begin();
315   }
316
317   void onWsEvent(AsyncWebSocket * server, AsyncWebSocketClient * client, AwsEventType
      type, void * arg, uint8_t *data, size_t len)
318   {
319     if(type == WS_EVT_CONNECT){
320       Serial.printf("ws[%s][%u] connect\n", server->url(), client->id());
321       client->printf("***> Hello Client %u :)", client->id());
322        ClientID = client->id();
323
324       client->ping();
325     } else if(type == WS_EVT_DISCONNECT){
326       Serial.printf("ws[%s][%u] disconnect\n", server->url(), client->id());
327     } else if(type == WS_EVT_ERROR){
328       Serial.printf("ws[%s][%u] error(%u): %s\n", server->url(), client->id(),
        *((uint16_t*)arg), (char*)data);
329     } else if(type == WS_EVT_PONG){
330       Serial.printf("ws[%s][%u] pong[%u]: %s\n", server->url(), client->id(), len,
        (len)?(char*)data:"");
331     } else if(type == WS_EVT_DATA){
332       AwsFrameInfo * info = (AwsFrameInfo*)arg;
333       String msg = "";
334       if(info->final && info->index == 0 && info->len == len){
335         //the whole message is in a single frame and we got all of it's data
336         Serial.printf("ws[%s][%u] %s-message[%llu]: ", server->url(), client->id(),
          (info->opcode == WS_TEXT)?"text":"binary", info->len);
337
338         if(info->opcode == WS_TEXT){
339           for(size_t i=0; i < info->len; i++) {
340             msg += (char) data[i];
341           }
342         } else {
343           char buff[3];
344           for(size_t i=0; i < info->len; i++) {
345             sprintf(buff, "%02x ", (uint8_t) data[i]);
346             msg += buff ;
347           }
348         }
349         Serial.printf("%s\n",msg.c_str());
350
351         if(info->opcode == WS_TEXT)
352           client->text("I got your text message");
353         else
```

```
354           client->binary("I got your binary message");
355       } else {
356         //message is comprised of multiple frames or the frame is split into multiple
          packets
357         if(info->index == 0){
358           if(info->num == 0)
359             Serial.printf("ws[%s][%u] %s-message start\n", server->url(),
              client->id(), (info->message_opcode == WS_TEXT)?"text":"binary");
360           Serial.printf("ws[%s][%u] frame[%u] start[%llu]\n", server->url(),
              client->id(), info->num, info->len);
361         }
362
363         Serial.printf("ws[%s][%u] frame[%u] %s[%llu - %llu]: ", server->url(),
          client->id(), info->num, (info->message_opcode == WS_TEXT)?"text":"binary",
          info->index, info->index + len);
364
365         if(info->opcode == WS_TEXT){
366           for(size_t i=0; i < len; i++) {
367             msg += (char) data[i];
368           }
369         } else {
370           char buff[3];
371           for(size_t i=0; i < len; i++) {
372             sprintf(buff, "%02x ", (uint8_t) data[i]);
373             msg += buff ;
374           }
375         }
376         Serial.printf("%s\n",msg.c_str());
377
378         if((info->index + len) == info->len){
379           Serial.printf("ws[%s][%u] frame[%u] end[%llu]\n", server->url(),
              client->id(), info->num, info->len);
380           if(info->final){
381             Serial.printf("ws[%s][%u] %s-message end\n", server->url(), client->id(),
              (info->message_opcode == WS_TEXT)?"text":"binary");
382             if(info->message_opcode == WS_TEXT)
383               client->text("I got your text message");
384             else
385               client->binary("I got your binary message");
386           }
387         }
388       }
389     }
390   }
391
392
393   /**********************************************************************************
394   * @brief  SetupServer() Webserver
395   * @return none
396   **********************************************************************************/
397
398   void SetupServer()
399    {
400     SPIFFS.begin();
401
402     ws.onEvent(onWsEvent);
403     server.addHandler(&ws);
404
405     events.onConnect([](AsyncEventSourceClient *client){
406      client->send( "hello!",NULL,millis(),1000);
407      });
408     server.addHandler(&events);
409
410
411
412
413
414   #ifdef ESP32
415     server.addHandler(new SPIFFSEditor(SPIFFS, http_username,http_password));
416   #elif defined(ESP8266)
417     server.addHandler(new SPIFFSEditor(http_username,http_password));
418   #endif
419
```

```
420
421     server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
422       request->send(SPIFFS, "/index.html" , String(), false, HTML_Processor);
423     });
424
425
426     server.serveStatic("/", SPIFFS, "/").setDefaultFile("index.html");
427     // Route to load style.css file
428     server.on("/style.css", HTTP_GET, [](AsyncWebServerRequest *request){
429       request->send(SPIFFS, "/style.css" , "text/css");
430     });
431
432
433     server.on("/java_script.js", HTTP_GET, [](AsyncWebServerRequest *request){
434       request->send(SPIFFS, "/java_script.js" , "text/javascript");
435     });
436
437     server.on("/java_script2.js", HTTP_GET, [](AsyncWebServerRequest *request){
438       request->send(SPIFFS, "/java_script2.js" , "text/javascript");
439     });
440
441     server.on("/color.jpg", HTTP_GET, [](AsyncWebServerRequest *request){
442       request->send(SPIFFS, "/color.jpg", "image/jpg");
443       });
444
445     server.on("/color1.png", HTTP_GET, [](AsyncWebServerRequest *request){
446       request->send(SPIFFS, "/color1.png", "image/png");
447       });
448
449     server.on("/color2.png", HTTP_GET, [](AsyncWebServerRequest *request){
450       request->send(SPIFFS, "/color2.png", "image/png");
451       });
452
453     server.on("/color3.png", HTTP_GET, [](AsyncWebServerRequest *request){
454       request->send(SPIFFS, "/color3.png", "image/png");
455       });
456
457
458        server.on("/favicon.ico", HTTP_GET, [](AsyncWebServerRequest *request){
459       request->send(SPIFFS, "/favicon.ico", "image/x-icon");
460        });
461
462
463
464     server.on("/color1", HTTP_GET, [] (AsyncWebServerRequest *request) {
465       String inputMessage1;
466       if (request->hasParam("rgb"))  {
467         inputMessage1 = request->getParam("rgb")->value();
468         }    else
469       inputMessage1 = "No message sent - rgb";
470
471      Serial.println(inputMessage1);
472
473    if (inputMessage1.startsWith("hex"))
474      ColorRGB16 = RGB32_2RGB16(inputMessage1);
475       request->send(200, "/buttons","OK" );
476     });
477
478
479   server.on("/color2", HTTP_GET, [] (AsyncWebServerRequest *request) {
480       String inputMessage1;
481       if (request->hasParam("rgba"))  {
482         inputMessage1 = request->getParam("rgba")->value();
483         }    else
484       inputMessage1 = "No message sent - rgba";
485
486      Serial.println(inputMessage1);
487
488
489       request->send(200, "/buttons","OK" );
490     });
491
492
```

```
493   server.on("/pattern", HTTP_GET, [] (AsyncWebServerRequest *request) {
494       String inputMessage1;
495       if (request->hasParam("simple"))  {
496         inputMessage1 = request->getParam("simple")->value();
497         sMatrixMessage.bTriggerPattern = true;
498         sMatrixMessage.Text = inputMessage1;
499         sWebSocket.bTriggerWebsocketMsg = true;
500
501         sWebSocket.WebsocketMsg = inputMessage1;
502
503         int num = inputMessage1.toInt();
504         sMatrixMessage.Pattern = num;
505
506      Serial.printf ("pattern ---- > %s -Trigger: %d--Color : 0x%x -- Bkg 0x%x ---
        Pattern : %d --- \n", sMatrixMessage.Text.c_str(), sMatrixMessage.bTriggerPattern,
        sMatrixMessage.Color, sMatrixMessage.BkgColor, sMatrixMessage.Pattern);
507
508
509         }    else
510       inputMessage1 = "No message sent - Simple";
511
512       request->send(200, "/buttons","[TestXYZ] Martin Michael " );
513     });
514
515
516   server.on("/neopat", HTTP_GET, [] (AsyncWebServerRequest *request) {
517       String inputMessage1;
518       if (request->hasParam("simple"))  {
519
520         inputMessage1 = request->getParam("simple")->value();
521         sMatrixMessage = GetMatrixTextValues(inputMessage1);
522         Serial.printf("input Msg : %s -Matrix- >> %s - %d - %d ",
          inputMessage1.c_str(), sMatrixMessage.Text.c_str(), sMatrixMessage.Color,
          sMatrixMessage.BkgColor);
523
524          sMatrixMessage.Pattern = 13; // Because we don't get it as Message
525
526            sMatrixMessage.bTriggerPattern = true;
527          sWebSocket.bTriggerWebsocketMsg = true;
528
529
530       sWebSocket.WebsocketMsg = inputMessage1;
531
532         }    else
533       inputMessage1 = "No message sent - Simple";
534
535       request->send(200, "/neopat","[Test] Das ist ein Test" );
536     });
537
538
539
540
541     server.onNotFound([](AsyncWebServerRequest *request){
542       Serial.printf("NOT_FOUND: ");
543       if(request->method() == HTTP_GET)
544         Serial.printf("GET");
545       else if(request->method() == HTTP_POST)
546         Serial.printf("POST");
547       else if(request->method() == HTTP_DELETE)
548         Serial.printf("DELETE");
549       else if(request->method() == HTTP_PUT)
550         Serial.printf("PUT");
551       else if(request->method() == HTTP_PATCH)
552         Serial.printf("PATCH");
553       else if(request->method() == HTTP_HEAD)
554         Serial.printf("HEAD");
555       else if(request->method() == HTTP_OPTIONS)
556         Serial.printf("OPTIONS");
557       else
558         Serial.printf("UNKNOWN");
559       Serial.printf(" http://%s%s\n", request->host().c_str(), request->url().c_str());
560
561       if(request->contentLength()){
```

```
562         Serial.printf("_CONTENT_TYPE: %s\n", request->contentType().c_str());
563         Serial.printf("_CONTENT_LENGTH: %u\n", request->contentLength());
564       }
565
566       int headers = request->headers();
567       int i;
568       for(i=0;i<headers;i++){
569         AsyncWebHeader* h = request->getHeader(i);
570         Serial.printf("_HEADER[%s]: %s\n", h->name().c_str(), h->value().c_str());
571       }
572
573       int params = request->params();
574       for(i=0;i<params;i++){
575         AsyncWebParameter* p = request->getParam(i);
576         if(p->isFile()){
577           Serial.printf("_FILE[%s]: %s, size: %u\n", p->name().c_str(),
           p->value().c_str(), p->size());
578         } else if(p->isPost()){
579           Serial.printf("_POST[%s]: %s\n", p->name().c_str(), p->value().c_str());
580         } else {
581           Serial.printf("_GET[%s]: %s\n", p->name().c_str(), p->value().c_str());
582         }
583       }
584
585       request->send(404);
586     });
587     server.onFileUpload([](AsyncWebServerRequest *request, const String& filename,
     size_t index, uint8_t *data, size_t len, bool final){
588       if(!index)
589         Serial.printf("UploadStart: %s\n", filename.c_str());
590       Serial.printf("%s", (const char*)data);
591       if(final)
592         Serial.printf("UploadEnd: %s (%u)\n", filename.c_str(), index+len);
593     });
594     server.onRequestBody([](AsyncWebServerRequest *request, uint8_t *data, size_t len,
     size_t index, size_t total){
595       if(!index)
596         Serial.printf("BodyStart: %u\n", total);
597       Serial.printf("%s", (const char*)data);
598       if(index + len == total)
599         Serial.printf("BodyEnd: %u\n", total);
600     });
601     server.begin();
602   }
603
604   /*****************************************************************************
605   * @brief  HTML_Processor(const String& var) For the HTML Webpage Values
606   * @param  var the Value to change
607   * @return String
608   *****************************************************************************/
609   String HTML_Processor(const String& var)
610   {
611     String Status = "";
612     Serial.print("---------- > ");
613     Serial.println(var);
614
615      if (var ==  "IP_ADRESSE")
616     Status = sIPAddress;
617
618    return (Status);
619   }
620
621
622   /*****************************************************************************
623   * @brief  GetRedish() in RGB16 565
624   * @param  none
625   * @return none
626   *****************************************************************************/
627   uint16_t GetRedish()
628   {
629       uint16_t Result;
630       Result = rand() % 0b11111;
631
```

```
    Result <<= 11;
     return (Result);
}

/*****************************************************************************
* @brief  GetGreenish() in RGB16 565
* @param  none
* @return none
*****************************************************************************/
uint16_t GetGreenish()
{
    uint16_t Result;
    Result = rand() % 0b11111;
    Result <<= 6;
    return (Result);
}

/*****************************************************************************
* @brief  GetBlueish() in RGB16 565
* @param  none
* @return none
*****************************************************************************/
uint16_t GetBlueish()
{
    uint16_t Result;
    Result = rand() % 0b11111;
    return (Result);
}



/*****************************************************************************
* @brief  NeoPixelTest() Just a Test with matrix
* @param  none
* @return none
*****************************************************************************/
void NeoPixelTest()
{

for (int y = 0; y < NeoPixelTileHeight; y++)
for (int x = 0; x < NeoPixelMatrixWidth; x++) {
matrix.drawPixel(x, y, ILI9341_WHITE);
matrix.show();
delay(Delay10);
ESP.wdtFeed();
matrix.drawPixel(x, y, ILI9341_BLACK );
matrix.show();

}

}


/*****************************************************************************
* @brief  NeoPixelPattern1() Just a Test with matrix
* @param  none
* @return none
*****************************************************************************/
void NeoPixelPattern1()
{

for (int y = 0; y < NeoPixelTileHeight; y++)
for (int x = 0; x <  NeoPixelMatrixWidth; x++) {
matrix.drawPixel(x, y, ILI9341_ORANGE );
matrix.show();
delay(Delay10);
ESP.wdtFeed();
}

for (int y = NeoPixelTileHeight; y >= 0; y--)
for (int x =  NeoPixelMatrixWidth; x >= 0; x--) {
matrix.drawPixel(x, y, ILI9341_BLACK );
matrix.show();
```

```
  delay(Delay10);

  }
  }

  /**************************************************************************
  * @brief  NeoPixelPattern2() Draw short Snake over the Screen
  * @param  none
  * @return none
  **************************************************************************/
  void NeoPixelPattern2()
  {

  for (int y = 0; y < NeoPixelTileHeight; y++)
  for (int x = 0; x <  NeoPixelMatrixWidth; x++) {
  matrix.drawPixel(x, y, ILI9341_RED );
  matrix.drawPixel(x+1, y, ILI9341_BLUE );
  matrix.drawPixel(x+2, y, ILI9341_BLUE );
  matrix.drawPixel(x+3, y, ILI9341_RED );
  matrix.show();
  delay(Delay10);
  ESP.wdtFeed();
  matrix.drawPixel(x, y, ILI9341_BLACK );
  matrix.drawPixel(x+1, y, ILI9341_BLACK );
  matrix.drawPixel(x+2, y, ILI9341_BLACK );
  matrix.drawPixel(x+3, y, ILI9341_BLACK );
  matrix.show();

  }
  }

  /**************************************************************************
  * @brief  NeoPixelPattern4() Fill Springle of Pixel
  * @param  none
  * @return none
  **************************************************************************/
  void NeoPixelPattern3()
  {
  int i = 0;
  while (i < 1000)
  {
  uint16_t x = rand() %  NeoPixelMatrixWidth;
  uint16_t y = rand() % NeoPixelTileHeight;

  uint16_t Color = rand() % 0x1fff;
  matrix.drawPixel(x, y, Color );
  matrix.show();
  delay(Delay10);
  ESP.wdtFeed();
  i++;
  }
  matrix.fillScreen(ILI9341_BLACK);
  }

  /**************************************************************************
  * @brief  NeoPixelPattern4() Draw Springle of Pixel
  * @param  none
  * @return none
  **************************************************************************/
  void NeoPixelPattern4()
  {
  int i = 0;
  while (i < 1000)
  {
  uint16_t x = rand() %  NeoPixelMatrixWidth;
  uint16_t y = rand() % NeoPixelTileHeight;

  uint16_t Color = rand() % 0xffff;
  matrix.drawPixel(x, y, Color );
  matrix.show();
  delay(10);
  ESP.wdtFeed();
  matrix.drawPixel(x, y, ILI9341_BLACK );
```

```
778   matrix.show();
779   i++;
780   }
781   }
782
783   /**************************************************************************
784   * @brief  NeoPixelPattern7() Draw Pulsing Red Screen
785   * @param  none
786   * @return none
787   **************************************************************************/
788   void NeoPixelPattern5()
789   {
790   int i = 0;
791   while (i < 500)
792   {
793   matrix.fillScreen( GetRedish());
794   delay(Delay10);
795   ESP.wdtFeed();
796   i++;
797   }
798   matrix.fillScreen(ILI9341_BLACK);
799   }
800
801   /**************************************************************************
802   * @brief  NeoPixelPattern7() Draw Pulsing Blue Screen
803   * @param  none
804   * @return none
805   **************************************************************************/
806   void NeoPixelPattern6()
807   {
808   int i = 0;
809   while (i < 500)
810   {
811   matrix.fillScreen( GetBlueish());
812   delay(Delay10);
813   ESP.wdtFeed();
814   i++;
815   }
816   }
817
818   /**************************************************************************
819   * @brief  NeoPixelPattern7() Draw Pulsing Green Screen
820   * @param  none
821   * @return none
822   **************************************************************************/
823   void NeoPixelPattern7()
824   {
825   int i = 0;
826   while (i < 500)
827   {
828   matrix.fillScreen( GetGreenish());
829   delay(Delay10);
830   ESP.wdtFeed();
831   i++;
832   }
833   }
834
835
836   /**************************************************************************
837   * @brief  NeoPixelPattern8() Draw Pulsing Red Rects
838   * @param  none
839   * @return none
840   **************************************************************************/
841   void NeoPixelPattern8()
842   {
843   for (int c = 0; c < 25; c++)
844   for (int i = 0; i < NeoPixelTileHeight; i++){
845   matrix.drawRect(i,i, NeoPixelMatrixWidth - i , NeoPixelTileHeight - i , GetRedish());
846   delay(Delay25);
847   ESP.wdtFeed();
848   matrix.show();
849   }
850   }
```

```
851
852
853
854   /*************************************************************************
855   * @brief  NeoPixelPattern9() Draw Pulsing Red Rects
856   * @param  none
857   * @return none
858   *************************************************************************/
859   void NeoPixelPattern9()
860   {
861
862   int counter = 0;
863
864   while (counter++ < 100)
865   {
866
867   for (int i = 0; i < NeoPixelTileHeight; i++){
868   matrix.drawRect(i,i, NeoPixelMatrixWidth - i , NeoPixelTileHeight - i , GetRedish());
869   delay(Delay10);
870   ESP.wdtFeed();
871   matrix.show();
872   }
873
874   matrix.fillScreen(ILI9341_BLACK);
875   for (int i = NeoPixelTileHeight; i>0; i--){
876   matrix.drawRect(i,i, NeoPixelMatrixWidth - i , NeoPixelTileHeight - i , GetRedish());
877   delay(Delay10);
878   ESP.wdtFeed();
879   matrix.show();
880   }
881
882   }
883   }
884
885   /*************************************************************************
886   * @brief  NeoPixelPattern10() Draw Lines
887   * @param  none
888   * @return none
889   *************************************************************************/
890   void NeoPixelPattern10()
891   {
892
893
894   for (int i = 0; i < 3; i++) {
895   for (int x = -4; x <  NeoPixelMatrixWidth +3; x++) {
896   matrix.drawFastVLine(x, 0, NeoPixelTileHeight, ILI9341_RED );
897   matrix.drawFastVLine(x+1, 0, NeoPixelTileHeight, ILI9341_BLUE );
898   matrix.drawFastVLine(x+2, 0, NeoPixelTileHeight, ILI9341_BLUE );
899   matrix.drawFastVLine(x+3, 0, NeoPixelTileHeight, ILI9341_RED );
900
901   matrix.show();
902   delay(Delay25);
903   ESP.wdtFeed();
904
905   matrix.drawFastVLine(x, 0, NeoPixelTileHeight, ILI9341_BLACK );
906   matrix.drawFastVLine(x+1, 0, NeoPixelTileHeight, ILI9341_BLACK );
907   matrix.drawFastVLine(x+2, 0, NeoPixelTileHeight, ILI9341_BLACK );
908   matrix.drawFastVLine(x+3, 0, NeoPixelTileHeight, ILI9341_BLACK );
909   matrix.show();
910   }
911
912
913   for (int x =  NeoPixelMatrixWidth +3; x > -4; x--) {
914   matrix.drawFastVLine(x, 0, NeoPixelTileHeight, ILI9341_RED );
915   matrix.drawFastVLine(x+1, 0, NeoPixelTileHeight, ILI9341_BLUE );
916   matrix.drawFastVLine(x+2, 0, NeoPixelTileHeight, ILI9341_BLUE );
917   matrix.drawFastVLine(x+3, 0, NeoPixelTileHeight, ILI9341_RED );
918
919   matrix.show();
920   delay(Delay25);
921   ESP.wdtFeed();
922
923   matrix.drawFastVLine(x, 0, NeoPixelTileHeight, ILI9341_BLACK );
```

```
matrix.drawFastVLine(x+1, 0, NeoPixelTileHeight, ILI9341_BLACK );
matrix.drawFastVLine(x+2, 0, NeoPixelTileHeight, ILI9341_BLACK );
matrix.drawFastVLine(x+3, 0, NeoPixelTileHeight, ILI9341_BLACK );
matrix.show();
} }

}

/*****************************************************************************
* @brief  NeoPixelPattern11() Draw Lines
* @param  none
* @return none
*****************************************************************************/
void NeoPixelPattern11()
{
for (int i = 0; i < 10; i++)  {
for (int x = 0; x <  NeoPixelMatrixWidth; x++)
{
matrix.drawFastVLine(x,0,NeoPixelTileHeight, rand() % 0xffff );
matrix.show();
delay(Delay25);
ESP.wdtFeed();
matrix.drawFastVLine(x,0,NeoPixelTileHeight, ILI9341_BLACK );
matrix.show();
}

for (int x =  NeoPixelMatrixWidth; x > 0; x--)
{
matrix.drawFastVLine(x,0,NeoPixelTileHeight, rand() % 0xffff );
matrix.show();
delay(Delay25);
ESP.wdtFeed();
matrix.drawFastVLine(x,0,NeoPixelTileHeight, ILI9341_BLACK );
matrix.show();
}
}}

/*****************************************************************************
* @brief  NeoPixelPattern12() Draw Lines
* @param  none
* @return none
*****************************************************************************/
void NeoPixelPattern12()
{

for (int i = 0; i < 5; i++)  {
for (int x = 0; x <  NeoPixelMatrixWidth; x++)
{
matrix.drawFastVLine(x,0,NeoPixelTileHeight, rand() % 0xffff );
matrix.show();
delay(Delay25);
ESP.wdtFeed();
}

for (int x =  NeoPixelMatrixWidth; x > 0; x--)
{
matrix.drawFastVLine(x,0,NeoPixelTileHeight, rand() % 0xffff );
matrix.show();
delay(Delay25);
}}}


/*****************************************************************************
* @brief  NeoPixelText1(const char Text[], uint16_t Color, uint16_t BkgColor)
* @param  Text to display
* @param  Color Text Color
* @param  BkgColor Background Color
* @return none
*****************************************************************************/
 void NeoPixelText1(const MatrixMessage_s Value)
{

  matrix.setTextColor(Value.Color,Value.BkgColor);
```

```
    strcpy(Buffer,Value.Text.c_str());
    int l = (strlen(Buffer) * 5 + NeoPixelMatrixWidth) *-1;

   for (int x =  NeoPixelMatrixWidth; x > l; x--)
   {
   matrix.fillScreen(ILI9341_BLACK);
    matrix.setCursor(x, 0);
    matrix.print(Buffer);
    matrix.show();
    delay(Delay100);
    ESP.wdtFeed();
   }

    matrix.fillScreen(ILI9341_BLACK);
  }


  /**************************************************************************************
  * @brief  HexStr2Int
  * @param  Value Hexstring
  * @return integer
  **************************************************************************************/
  uint32_t HexStr2Int(String Value)
  {
      uint32_t Result = 0;
      char* p;
      Result = strtol(Value.c_str(), &p, 16);
          return (Result);
    }


  /**************************************************************************************
  * @brief  GetMatrixTextValues
  * @param  Value String from index HTML
  * @return struct
  **************************************************************************************/
  MatrixMessage_s GetMatrixTextValues(String Value)
  {
  MatrixMessage_s Result;
  int  num;
  String S;
  uint16_t d1,d2;
  if (Value.startsWith("[Text]") )
  {

   num = Value.indexOf("=");
   Value.remove(0, num + 1);
   Value.trim();

   num = Value.indexOf("[Color]");
   Result.Text = Value.substring(0, num);
   Result.Text.trim();
   num = Value.indexOf("=");
   Value.remove(0,num + 1);
   num = Value.indexOf("[Bkg]");
   S = Value.substring(0, num);
   S.trim();
   d1 =  RGB32_2RGB16(S);
   Result.Color = d1;

  num = Value.indexOf("=");
  Value.remove(0,num +1);
  Value.trim();
  d2 = RGB32_2RGB16(Value);
  Result.BkgColor = d2;
  Serial.printf( "RGB16 - %d - %d", d1, d2);

  }
  return Result;
  }
```