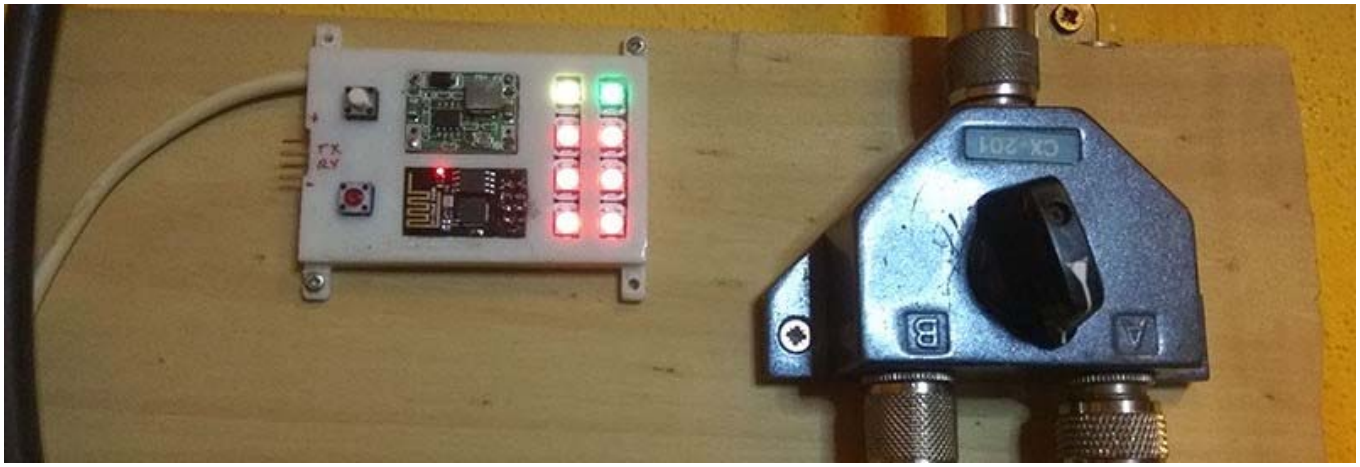
ESP8266 SolarData-RGB



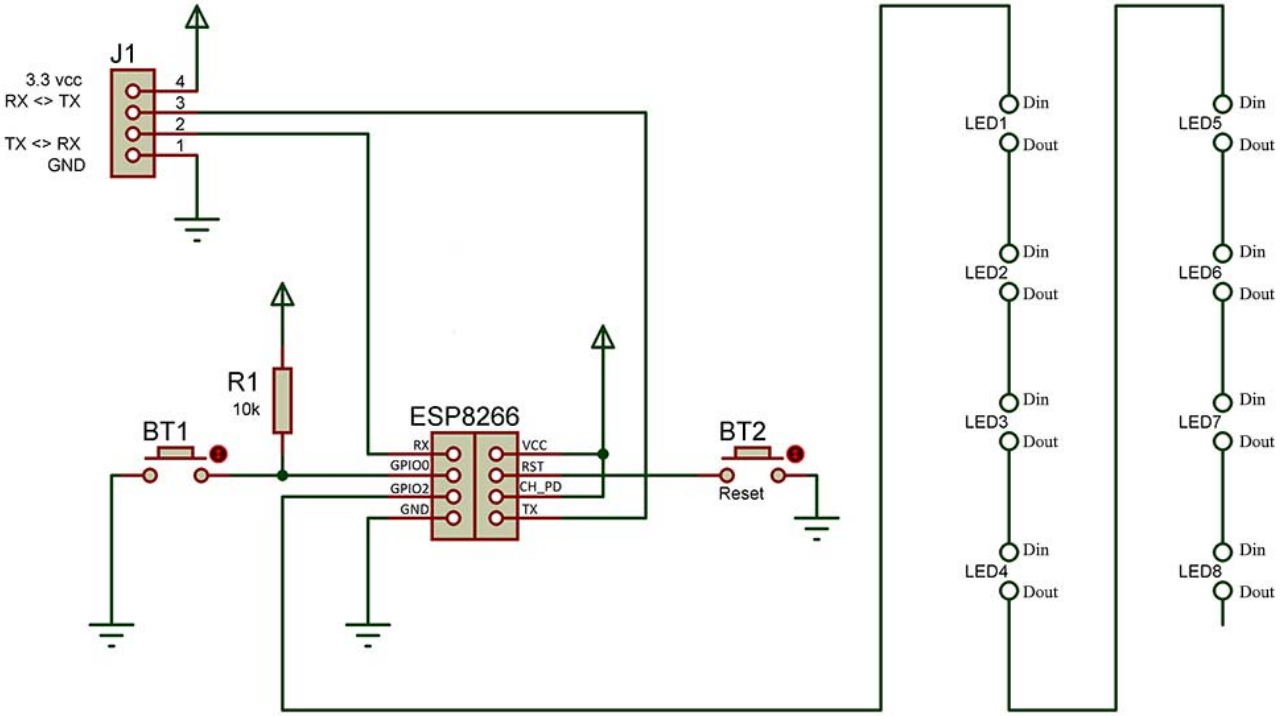
Se trata de un sencillo circuito que nos muestra en una tira LED WS2812B, por medio de colores, el estado de las bandas HF de radioaficionados, obteniendo dicha información desde la página web hamqsl.com.

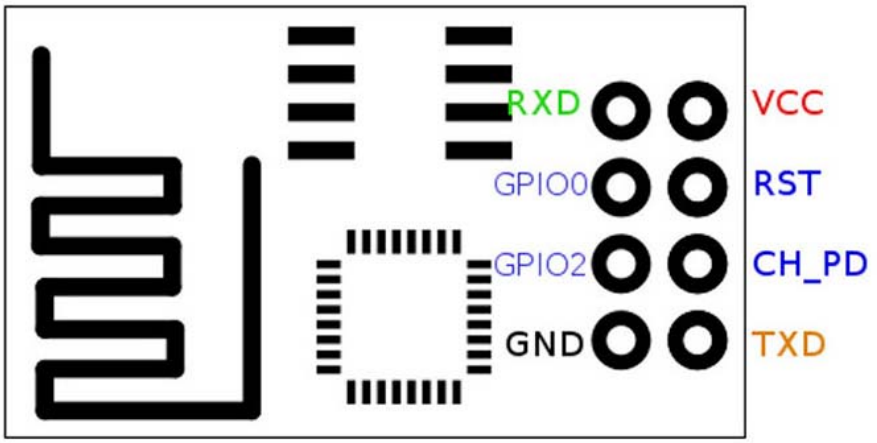
Para el control y conexión a la red wifi, se utiliza el módulo ESP8266, utilizado en este circuito el primer modelo ESP-01, que es el mas simple y pequeño de la serie pero suficiente para este proyecto.

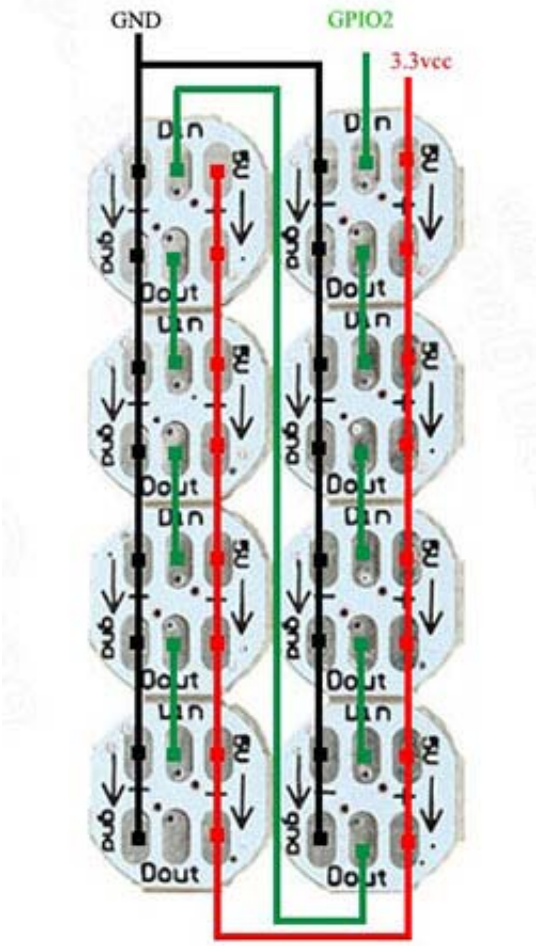
Material

- Modulo ESP8266 ESP-01.
- 8 LEDs RGB WS2812, tipo neopixel o similares.
- 2 pulsadores NA (normalmente abiertos)
- 1 resistencia de 10K.
- Conversor USB a TTL de 3.3v (solo para la programación)
- Fuente de alimentación de 3.3v para su instalación definitiva.

Circuito


J1
3.3 vcc
RX <> TX
TX <> RX
GND
4
3
2
1
R1
10k
BT1
ESP8266
RX
GPIO0
GPIO2
GND
VCC
RST
CH_PD
TX
BT2
Reset
LED1
LED2
LED3
LED4
LED5
LED6
LED7
LED8
Din
Dout


RXD
GPIO0
GPIO2
GND
VCC
RST
CH_PD
TXD



Programación

1. Instalar la última versión de Arduino.
2. Instalar la tarjeta del ESP8266.
   - Abrir la aplicación de aplicación de Arduino e ir a Preferencias desde el menú Archivo.
   - Introducir `http://arduino.esp8266.com/stable/package_esp8266com_index.json` en "Gestor de URLs Adicionales de Tarjetas" (puede introducir varias direcciones separandolas por comas ',').
   - Abrir *Gestor de tarjetas* desde el menú Herramientas/Placa. Buscamos por ESP8266 e instalamos.
   - Podéis encontrar mas información u otros modos de instalación en su página de GitHub.
3. Instalación de la librería que controla los LED's.
   - Desde el menú Programa/Incluir Librería, abrir *Gestionar Librerías*. Buscar e instalar la librería llamada **Adafruit NeoPixel** by Adafruit.
4. Seleccionar el módulo *Generic ESP8266 Module* para el ESP-01; seleccionar el puerto COM de su conversor USB-TTL; en *Upload speed* seleccionar 115200.
5. Para subir el programa al ESP8266 ESP-01, primero debemos iniciarlo en modo programación, para ello, siguiendo el circuito propuesto:
   - Conectar la alimentación, Tx y Rx a nuestro USB-TTL **(importante que éste sea de 3.3v, tanto alimentación como bus de datos)**.
   - Pulsar y mantener pulsado el pulsador de reset "BT2".
   - Pulsar y mantener pulsado el pulsador 1 "BT1".

- Soltar pulsador de reset "BT2".
- Esperar 2 segundos y soltar el pulsador 1 "BT1".
- Subir el programa desde la aplicación de Arduino.

Se utiliza la librería ***WiFiManager*** incluida directamente en el proyecto, por lo que no hace falta sea instalada. Se ha realizado de este modo ya que ésta es la que nos crea la web de configuración que por defecto viene en inglés. La incluida, esta modificada para que la web esté en español.

Para mas información sobre ***WiFiManager***, acceder a su página de GitHub.

Funcionamiento y puesta en marcha.

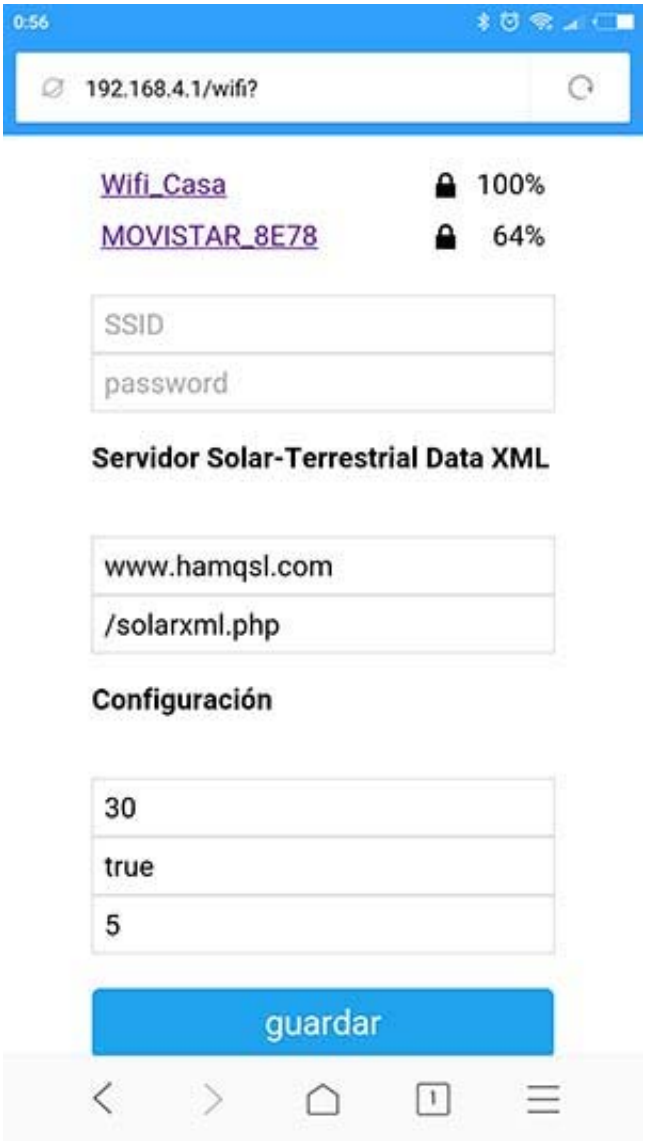
Una vez subido el *sketch* a nuestro ESP8266 y si todo ha ido bien, deberíamos ver como se iluminan todos los leds en azul durante unos segundos pasando a continuación a un color morado.

El color morado, significa que nuestro circuito a entrado en modo AP (*"Acces Point" o "Punto de Acceso"*), creando una red Wifi propia con el nombre AP_SolarData.

Condigurar nuestra red wifi y otras opciones.

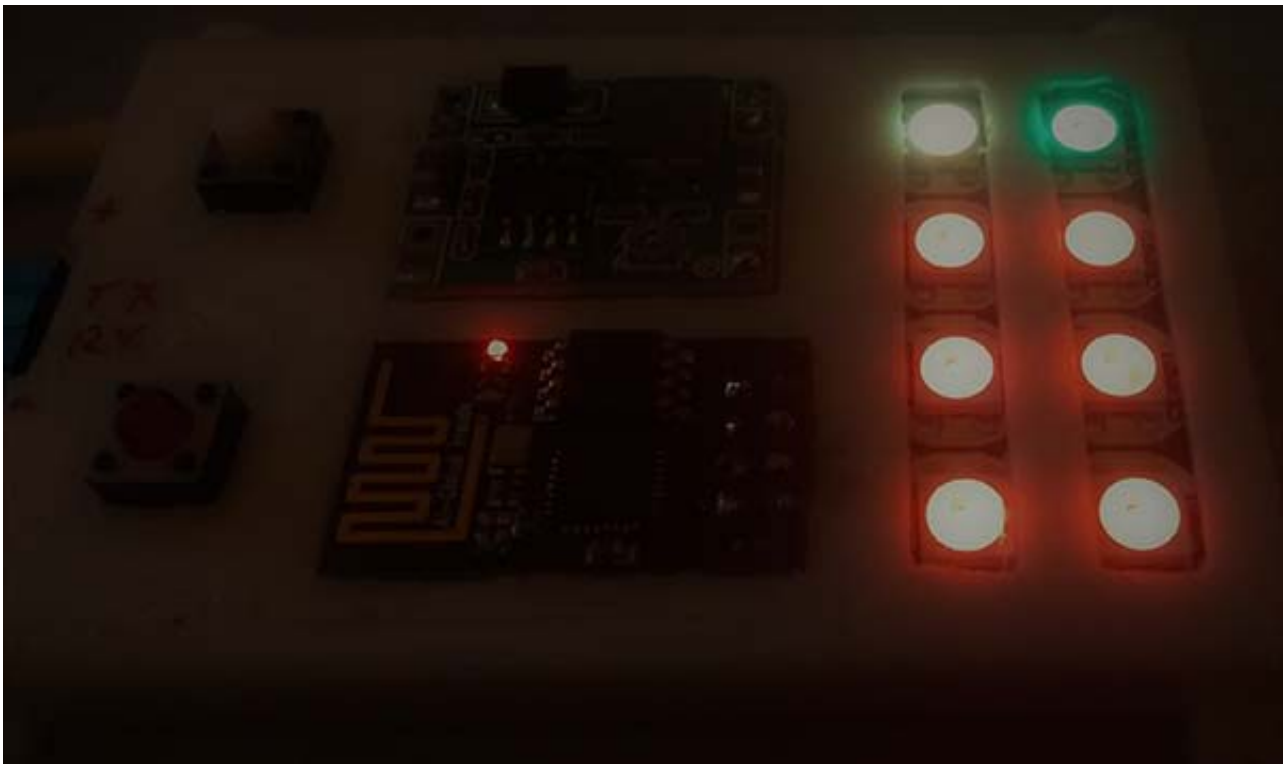
- Conectarse con una tablet, pc, teléfono o similar a la red wifi creada *AP_SolarData*, es una red abierta.
- Abrir un navegador web, en caso de no redireccionarnos directamente, acceder a la dirección 192.168.4.1
- En la siguiente pantalla, pulsar en *Configurar WIFI*.

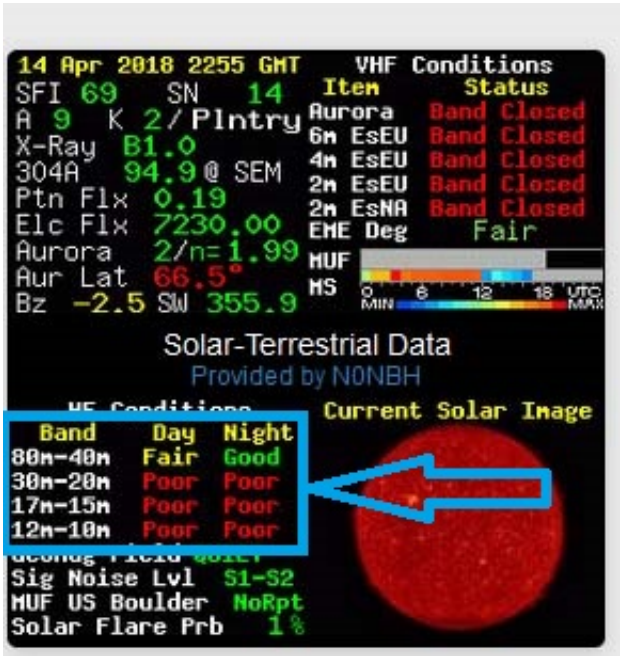
- Nos aparecerá una página como la siguiente:



- En el listado, pulsar sobre nuestra red e introducir la contraseña de acceso.
- El apartado *Servidor Solar-Terrestrial Data XML* son el host y la url de acceso al XML que nos sirve los datos, no deberían modificarse salvo que cambie la ubicación del archivo.
- En el apartado *Configuración*, podemos seleccionar algunos parámetros a nuestro gusto:
  - El Primero (30) es la intensidad de iluminación de los LEDs, podemos poner un valor comprendido entre 0 y 255.
  - El segundo nos permite escribir *true* o *false*
    - **true**: Se apagarán los LEDs transcurridos unos segundos. Pulsando el "BT2" se realiza una nueva solicitud de datos a la web y reilumina los leds según correspondan.
    - **false**: Se mantienen los LEDs encendidos. Pulsando el "BT2" se realiza una nueva solicitud de datos a la web y reilumina los leds según correspondan.
    - En cualquier caso, el módulo se desconecta de la red y pasa a bajo consumo una vez cargados los datos.
  - El tercer apartado nos permite configurar (en caso de ser true el apartado anterior) los segundos que se mantendrán los LED's iluminados.
- Pulsar sobre *guardar* y si todo ha ido bien nuestro ESP debería reiniciarse, conectarse a nuestra wifi y mostrar los colores según corresponda. Si la iluminación de los LEDs sigue siendo morada, significa que no ha conseguido conectarse a nuestra wifi, por lo que habrá

entrado en modo AP para volver a configurar.





Cambiar configuración.

Una vez nuestro ESP tiene los datos guardados y consigue entrar a nuestra wifi, éste no vuelve a ponerse en modo AP, solo se activa este modo si no es capaz de conectarse a nuestra wifi o si reiniciamos los datos.

Para hacer un reinicio de datos para volver a entrar en modo configuración hay que:

- Reiniciar el ESP con el pulsador 2 "BT2" y soltarlo.
- Cuando la iluminación está en AZUL, pulsar el BT1.

Ahora debería iluminarse en morado, ya está listo para volver a configurar desde la wifi *AP_SolarData*.

Iluminación AZUL es una ventana de 2,5 segundos que nos permite reiniciar los valores a nuestro ESP pulsando el BT1

Mas información en la web de GitHub: https://github.com/DonJaume/ESP8266_SolarData_RGB

```
// NOMBRE DEL ARCHIVO     ESP8266_SolarData_RGB.ino



#include <ESP8266WiFi.h>          //https://github.com/esp8266/Arduino
#include <DNSServer.h>
#include <ESP8266WebServer.h>
#include "WiFiManager.h"          //https://github.com/tzapu/WiFiManager
#include <Adafruit_NeoPixel.h>
#include <EEPROM.h>


#define TRIGGER_PIN 0               //pin utilizado para resetear los valores de por defecto. Con esto se inicia el asistente de
configuración. (debe pulsarse cuando los LED están en azul)
#define PIN 2                       //pin de salida donde se conecta el bus de información de los neopixels (LED's)
#define NUMPIXELS 8                 //numero de LED's conectados a la matriz
#define TIEMPO_RESET 2500           //tiempo en milisegundos del tiempo de ventana para resetear los valores por defecto.

#define L_Good      0x9600          //Verde
#define L_Poor      0x960000        //Rojo
#define L_Fair      0x969600        //Amarillo
#define L_Reset     0x96            //Azul
#define L_Off       0x0             //Apagado
#define L_Program   0x892496        //Morado

Adafruit_NeoPixel pixels = Adafruit_NeoPixel(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);


//------------------------------------------------------------ VARIABLES GLOBALES
-------------------------------------------------------------
char host[20] = "www.hamqsl.com";   //dir 0   host por defecto al que se va a conectar
char Url[20] = "/solarxml.php";     //dir 20  direccion por defecto del host donde hacemos la petición del XML
char Tiempo[10] = "5";              //dir 40  tiempo en segundos por defecto para apagar los leds (si apagar es false no se tiene en
cuenta)
char Apagar[5] = "true";            //dir 50  establece si se apagarán los LED tras un periodo de tiempo
char Brillo[3] = "30";              //dir 55  intensidad de iluminación de los LED comprendido en tre 0 y 255

char Estado[8];                     //variable usada para almacenar "G","F","P" (Good, Fair, Poor) correlativamente segun nos los
proporciona el host.
char brillo = 30;                   //variable convertida a información interpretable de la variable Brill[3] declarada anteriormente
int TmpApagado = 5000;              //variable convertida a tipo int multiplicada por 1000, tiempo tras apadado en milisegundos
bool apagar = true;                 //variable convertida a tipo booeana

bool shouldSaveConfig = false;      //Bandera para guardar los datos

//------------------------------------------------------------ Llamada activa el guardado de datos
-------------------------------------------------------------
void saveConfigCallback () {
```

```
  shouldSaveConfig = true;
}

//------------------------------------------------------------ Llamada antes de entrar en modo AP
-------------------------------------------------------------
void configModeCallback(WiFiManager *myWiFiManager){
  allColor(L_Program);
  Serial.println(myWiFiManager->getConfigPortalSSID());
}

//------------------------------------------------------------ Función para grabar en la EEPROM
-------------------------------------------------------------
void grabar(int addr, char* inchar, int tamano) {
  for (int i = 0; i < tamano; i++) {
    if(inchar[i] == 0) EEPROM.write(addr+i, 255);
    else EEPROM.write(addr+i, inchar[i]);
  }
  EEPROM.commit();
  delay(20);
}

//------------------------------------------------------------ Función para leer la EEPROM
-------------------------------------------------------------
void leer(int addr, int tm, char* lect)
{
   byte lectura;
   int e = 0;
   for (int i = addr; i < (addr + tm); i++)
   {
      lectura = EEPROM.read(i);
      if (lectura != 255) lect[e] = (char)lectura;
      else lect[e] = 0;
      e++;
   }
}

//------------------------------------------------------------ CARGAR DATOS DE LA EEPROM
-------------------------------------------------------------
void cargaDatos()
{
  leer(0,20, host);
  leer(20,20, Url);
  leer(40,10, Tiempo);
  leer(50,5, Apagar);
  leer(55,3, Brillo);

  refrescaDatos();
}
```

```
//------------------------------------------------------------ CONVIERTE LOS DATOS A SUS VARIABLES UTILIZABLES
------------------------------------------------------------
void refrescaDatos()
{
  String temp="";

  for(int i= 0; i<10; i++)
    if(Tiempo[i] != 0) temp+= Tiempo[i];
  TmpApagado = temp.toInt() * 1000;

  temp="";
  for(int i= 0; i<3; i++)
    if(Brillo[i] != 0) temp+= Brillo[i];
  brillo = (char)temp.toInt();

  if(Apagar[0] == 'f') apagar = false;
  else if(Apagar[0] == 'F') apagar = false;
  else apagar = true;

  Serial.println();
  Serial.println("Carga de datos:");
  Serial.print("Host------> "); Serial.println(host);
  Serial.print("Url-------> "); Serial.println(Url);
  Serial.print("Tiempo----> "); Serial.println(Tiempo);
  Serial.print("Apagar----> "); Serial.println(Apagar);
  Serial.print("Brillo----> "); Serial.println(Brillo);
}

//------------------------------------------------------------ FUNCION "SPLIT"
------------------------------------------------------------
//Busca los estados de propagación en el texto recibido del XML consultado en la web
//introduce en la matriz de 8 bytes los caracteres 'G','F','P' segun los estados GOOD, FAIR o POOR respectivamente
bool Split(String texto)
{
  bool retorno = false;
  int puntero = 0;
  for(int i =0; i < 4; i++)
  {
    puntero = texto.indexOf("\"day\">", puntero + 1);
    if(puntero < 0) break;
    Estado[i] = texto[puntero + 6];
    retorno = true;
  }
  puntero = 0;
  for(int i =0; i < 4; i++)
  {
    puntero = texto.indexOf("\"night\">", puntero + 1);
    if(puntero < 0) break;
    Estado[i+4] = texto[puntero + 8];
```

```
  }
  return retorno;
}

//------------------------------------------------------------ FUNCIÓN TRADUCTOR DE CARÁCTERES A COLORES
//-------------------------------------------------------------
//traduce los caracteres 'G','F','P' a los colores correspondientes
uint32_t SetColor (char est)
{
  if(est == 'G') return L_Good;      //Verde
  if(est == 'F') return L_Fair;  //Amarillo
  if(est == 'P') return L_Poor;     //Rojo
  return pixels.Color(0,0,0);
}

//------------------------------------------------------------ FUNCIÓN PINTA TODOS LOS LEDS
//-------------------------------------------------------------
void allColor(uint32_t _color)
{
  for(int i=0; i< NUMPIXELS; i++)
    pixels.setPixelColor(i, _color);
  pixels.show();
}

//------------------------------------------------------------ SECUENCIA DE APAGADO
//-------------------------------------------------------------
void Sec_Apagado()
{
  delay(TmpApagado);              //espera el tiempo programado

        for(int i=0; i< 4; i++)         //realizamos 4 parpadeos de los leds
        {
          pixels.setBrightness(10);
          pixels.show();
          delay(500);
          pixels.setBrightness(brillo);
          pixels.show();
          delay(500);
          Serial.print("*");
        }
        allColor(L_Off);            //apagamos todos los leds
}

//-------------------------------------------------------------------------------------------------------------------------
//------------------------------------------------------------ SETUP ------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------------
void setup() {

  Serial.begin(115200);                 //iniciamos puerto serie a 115200
```

```
Serial.println();

pinMode(TRIGGER_PIN, INPUT);          //pin TRIGGER como entrada
pixels.begin();                       //iniciamos la libreria de los neopixels
pixels.setBrightness(brillo);         //asigna el la intensidad de brillo de los led's
allColor(L_Off);                      //apaga todos los LED's

EEPROM.begin(256);                    //iniciamos la libreria de manejo de la EEPROM



//Creamos las variables para los nuevos parámetros que serán visualizados en la web de configuración
WiFiManagerParameter custom_text0("<p><strong> Servidor Solar-Terrestrial Data XML </p> </strong>");
WiFiManagerParameter custom_xml_host("host", "host XML", host, 21);
WiFiManagerParameter custom_xml_url("url", "url XML", Url, 21);
WiFiManagerParameter custom_text1("<p><strong> Configuración </p> </strong>");
WiFiManagerParameter custom_Brillo("Brillo", "Brillo 0-255", Brillo, 4);
WiFiManagerParameter custom_Apagar("Apagar", "Apagar true or false", Apagar, 6);
WiFiManagerParameter custom_Tiempo("Tiempo", "Tiempo apagado en segundos", Tiempo, 11);

WiFiManager wifiManager;              //iniciamos la libreria del mánager Wifi

//Seteamos la llamada a la función de guardado de datos
wifiManager.setSaveConfigCallback(saveConfigCallback);

//Seteamos la llamada a función antes de entrar en modo AP
wifiManager.setAPCallback(configModeCallback);


//Introducimos los parametros personalizados de la web
wifiManager.addParameter(&custom_text0);
wifiManager.addParameter(&custom_xml_host);
wifiManager.addParameter(&custom_xml_url);
wifiManager.addParameter(&custom_text1);
wifiManager.addParameter(&custom_Brillo);
wifiManager.addParameter(&custom_Apagar);
wifiManager.addParameter(&custom_Tiempo);



//******************************** VENTANA PARA RESETEAR A VALORES DE FABRICA ****************
allColor(L_Reset);                                                  //ponemos todos los led en azul
delay(TIEMPO_RESET);                                               //esperamos el tiempo asignado

if ( digitalRead(TRIGGER_PIN) == LOW ) wifiManager.resetSettings();   //si el botón está pulsado reseteamos valores por defecto
else
{
  cargaDatos();                                                     //sino cargamos los valores guardados en la EEPROM
  pixels.setBrightness(brillo);                                     //seteamos el brillo de los led's según la configuración
```

```
    guardada
  }

  allColor(L_Off);                                                    //apagamos todos los led's

  wifiManager.setDebugOutput(false);                                   //modo DEBUG de wifimanager apagado

  //*****************************************FIN VENTANA **********************************

  //Loop del mánager
  //Si no consigue conectarse a la Wifi o hemor realizado un reinicio de valores de "fabrica" crea un Acces Point con la web de
  configuración
  //debe redireccionar automaticamente, en todo caso su dirección es 192.168.4.1
  if (!wifiManager.autoConnect("AP_SolarData")) {
    Serial.println("Fallo de conexión");
    delay(3000);
    //Si los datos de conexíon no son correctos resetea el ESP
    ESP.reset();
    delay(5000);
  }

  //Wifi conectada
  Serial.println();
  Serial.println("Conectado:");

  //Leemos los datos devueltos de la página de configuración y los asignamos a sus variables
  strcpy(host, custom_xml_host.getValue());
  strcpy(Url, custom_xml_url.getValue());
  strcpy(Brillo, custom_Brillo.getValue());
  strcpy(Apagar, custom_Apagar.getValue());
  strcpy(Tiempo, custom_Tiempo.getValue());

  //Salvamos parametros personalizados
  if (shouldSaveConfig) {
    Serial.println("Guardando configuración");

    grabar(0, host, 20);
    grabar(20, Url, 20);
    grabar(40, Tiempo, 10);
    grabar(50, Apagar, 5);
    grabar(55, Brillo, 3);

    refrescaDatos();
  }

  Serial.print("Ip local: ");
  Serial.println(WiFi.localIP());

}
```

```
//-------------------------------------------------------------------------------------------------------------------------------
-------
//------------------------------------------------------------------ LOOP
--------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------------------
-------
void loop() {
   //Conexion al host ************************************************

    Serial.println("");
    Serial.print("Conectando a ");
    Serial.println(host);

    WiFiClient client;                              //Creamos el cliente de conexión TCP
    const int httpPort = 80;                        //puerto a usar
    if (!client.connect(host, httpPort)) {          //intentamos conectar al host y puertos asignados
      Serial.println("Conexión fallida");
      return;
    }


    String url(Url);                                //Convertimos a string la url de la petición XML
    Serial.print("Solicitando URL: ");
    Serial.println(url);

    // Enviamos la petición al servidor
    client.print(String("GET ") + url + " HTTP/1.1\r\n" +
                 "Host: " + host + "\r\n" +
                 "Connection: close\r\n\r\n");
    unsigned long timeout = millis();

    while (client.available() == 0) {               //esperamos respuesta del servidor
        if (millis() - timeout > 5000) {
        Serial.println(">>> Tiempo de espera del cliente!");
        client.stop();                              //si pasa el tiempo de espera cerramos la conexión
        return;
      }
    }

    while(client.available()){                      //si tenemos respuesta del servidor
      String line = client.readStringUntil('\r');   //leemos las lineas enviadas
      if(Split(line))
      {
        client.stop();                                //las enviamos al decodificador para obtener los datos requeridos
        Serial.print(line);                           //imprimimos las lineas por el puerto serie
        break;
```

```
    }
  }

  Serial.println();
  Serial.println("Cerrando conexión...");

  //Seteamos los colores de los pixels según los datos obtenidos del XML
  for(int i=0; i < NUMPIXELS; i++) pixels.setPixelColor(i, SetColor(Estado[i]));
  pixels.show();


  if(apagar)  Sec_Apagado();                        //Si en la configuración está seteado el apagado de los leds


  Serial.println("Durmiendo ESP...");
  ESP.deepSleep(0);                       //Una vez finalizada la consulta, se apaguen o no los LED pasamos el ESP en modo de bajo
  consumo
                                          //solo despierta al resetearlo
}
```

```
//NOMBRE DEL ARCHIVO            WiFiManager.h



/**************************************************************
   WiFiManager is a library for the ESP8266/Arduino platform
   (https://github.com/esp8266/Arduino) to enable easy
   configuration and reconfiguration of WiFi credentials using a Captive Portal
   inspired by:
   http://www.esp8266.com/viewtopic.php?f=29&t=2520
   https://github.com/chriscook8/esp-arduino-apboot
   https://github.com/esp8266/Arduino/tree/master/libraries/DNSServer/examples/CaptivePortalAdvanced
   Built by AlexT https://github.com/tzapu
   Licensed under MIT license
 **************************************************************/

#ifndef WiFiManager_h
#define WiFiManager_h

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <DNSServer.h>
#include <memory>

extern "C" {
  #include "user_interface.h"
}

const char HTTP_HEAD[] PROGMEM            = "<!DOCTYPE html><html lang=\"en\"><head><meta name=\"viewport\"
content=\"width=device-width, initial-scale=1, user-scalable=no\"/><title>{v}</title>";
const char HTTP_STYLE[] PROGMEM           = "<style>.c{text-align: center;} div,input{padding:5px;font-size:1em;} input{width:95%;}
body{text-align: center;font-family:verdana;}
button{border:0;border-radius:0.3rem;background-color:#1fa3ec;color:#fff;line-height:2.4rem;font-size:1.2rem;width:100%;} .q{float:
right;width: 64px;text-align: right;} .l{background:
url(\"
g4eFydHTCjaf3AAAAZElEQVQ4je2NSw7AIAhEBamKn97/uMXEGBvozkWb9C2Zx4xzWykBhFAeYp9gkLyZE0zIMno9n4g19hmdY39scwqVkOXaxph0ZCXQcqxSpgQpONa59wkRDO
L93eAXvimwlbPbwwVAegLS1HGfZAAAAABJRU5ErkJggg==\") no-repeat left center;background-size: 1em;}</style>";
const char HTTP_SCRIPT[] PROGMEM          = "<script>function
c(l){document.getElementById('s').value=l.innerText||l.textContent;document.getElementById('p').focus();}</script>";
const char HTTP_HEAD_END[] PROGMEM        = "</head><body><div style='text-align:left;display:inline-block;min-width:260px;'>";
const char HTTP_PORTAL_OPTIONS[] PROGMEM  = "<form action=\"/wifi\" method=\"get\"><button>Configurar WiFi</button></form><br/><form
action=\"/0wifi\" method=\"get\"><button>Configurar WiFi (Sin escaneo)</button></form><br/><form action=\"/i\"
method=\"get\"><button>Info</button></form><br/><form action=\"/r\" method=\"post\"><button>Reset</button></form>";
const char HTTP_ITEM[] PROGMEM            = "<div><a href='#p' onclick='c(this)'>{v}</a> <span class='q {i}'>{r}%</span></div>";
const char HTTP_FORM_START[] PROGMEM      = "<form method='get' action='wifisave'><input id='s' name='s' length=32
placeholder='SSID'><br/><input id='p' name='p' length=64 type='password' placeholder='password'><br/>";
const char HTTP_FORM_PARAM[] PROGMEM      = "<br/><input id='{i}' name='{n}' maxlength={l} placeholder='{p}' value='{v}' {c}>";
const char HTTP_FORM_END[] PROGMEM        = "<br/><button type='submit'>guardar</button></form>";
const char HTTP_SCAN_LINK[] PROGMEM       = "<br/><div class=\"c\"><a href=\"/wifi\">Escanear</a></div>";
```

```
const char HTTP_SAVED[] PROGMEM          = "<div>Credenciales guardadas<br />Intentando conectar ESP a la red.<br />Si falla, vuelva
a conectarse a AP para volver a intentarlo</div>";
const char HTTP_END[] PROGMEM            = "</div></body></html>";

#define WIFI_MANAGER_MAX_PARAMS 10

class WiFiManagerParameter {
  public:
    WiFiManagerParameter(const char *custom);
    WiFiManagerParameter(const char *id, const char *placeholder, const char *defaultValue, int length);
    WiFiManagerParameter(const char *id, const char *placeholder, const char *defaultValue, int length, const char *custom);

    const char *getID();
    const char *getValue();
    const char *getPlaceholder();
    int         getValueLength();
    const char *getCustomHTML();
  private:
    const char *_id;
    const char *_placeholder;
    char       *_value;
    int         _length;
    const char *_customHTML;

    void init(const char *id, const char *placeholder, const char *defaultValue, int length, const char *custom);

    friend class WiFiManager;
};


class WiFiManager
{
  public:
    WiFiManager();

    boolean       autoConnect();
    boolean       autoConnect(char const *apName, char const *apPassword = NULL);

    //if you want to always start the config portal, without trying to connect first
    boolean       startConfigPortal();
    boolean       startConfigPortal(char const *apName, char const *apPassword = NULL);

    // get the AP name of the config portal, so it can be used in the callback
    String        getConfigPortalSSID();

    void          resetSettings();

    //sets timeout before webserver loop ends and exits even if there has been no setup.
    //useful for devices that failed to connect at some point and got stuck in a webserver loop
```

```
  //in seconds setConfigPortalTimeout is a new name for setTimeout
  void          setConfigPortalTimeout(unsigned long seconds);
  void          setTimeout(unsigned long seconds);

  //sets timeout for which to attempt connecting, useful if you get a lot of failed connects
  void          setConnectTimeout(unsigned long seconds);


  void          setDebugOutput(boolean debug);
  //defaults to not showing anything under 8% signal quality if called
  void          setMinimumSignalQuality(int quality = 8);
  //sets a custom ip /gateway /subnet configuration
  void          setAPStaticIPConfig(IPAddress ip, IPAddress gw, IPAddress sn);
  //sets config for a static IP
  void          setSTAStaticIPConfig(IPAddress ip, IPAddress gw, IPAddress sn);
  //called when AP mode and config portal is started
  void          setAPCallback( void (*func)(WiFiManager*) );
  //called when settings have been changed and connection was successful
  void          setSaveConfigCallback( void (*func)(void) );
  //adds a custom parameter
  void          addParameter(WiFiManagerParameter *p);
  //if this is set, it will exit after config, even if connection is unsuccessful.
  void          setBreakAfterConfig(boolean shouldBreak);
  //if this is set, try WPS setup when starting (this will delay config portal for up to 2 mins)
  //TODO
  //if this is set, customise style
  void          setCustomHeadElement(const char* element);
  //if this is true, remove duplicated Access Points - defaut true
  void          setRemoveDuplicateAPs(boolean removeDuplicates);

private:
  std::unique_ptr<DNSServer>        dnsServer;
  std::unique_ptr<ESP8266WebServer> server;

  //const int     WM_DONE                 = 0;
  //const int     WM_WAIT                 = 10;

  //const String  HTTP_HEAD = "<!DOCTYPE html><html lang=\"en\"><head><meta name=\"viewport\" content=\"width=device-width,
  initial-scale=1\"/><title>{v}</title>";

  void          setupConfigPortal();
  void          startWPS();

  const char*   _apName                 = "no-net";
  const char*   _apPassword             = NULL;
  String        _ssid                   = "";
  String        _pass                   = "";
  unsigned long _configPortalTimeout    = 0;
  unsigned long _connectTimeout         = 0;
```

```
unsigned long _configPortalStart      = 0;

IPAddress     _ap_static_ip;
IPAddress     _ap_static_gw;
IPAddress     _ap_static_sn;
IPAddress     _sta_static_ip;
IPAddress     _sta_static_gw;
IPAddress     _sta_static_sn;

int           _paramsCount            = 0;
int           _minimumQuality         = -1;
boolean       _removeDuplicateAPs     = true;
boolean       _shouldBreakAfterConfig = false;
boolean       _tryWPS                 = false;

const char*   _customHeadElement      = "";

//String        getEEPROMString(int start, int len);
//void          setEEPROMString(int start, int len, String string);

int           status = WL_IDLE_STATUS;
int           connectWifi(String ssid, String pass);
uint8_t       waitForConnectResult();

void          handleRoot();
void          handleWifi(boolean scan);
void          handleWifiSave();
void          handleInfo();
void          handleReset();
void          handleNotFound();
void          handle204();
boolean       captivePortal();
boolean       configPortalHasTimeout();

// DNS server
const byte    DNS_PORT = 53;

//helpers
int           getRSSIasQuality(int RSSI);
boolean       isIp(String str);
String        toStringIp(IPAddress ip);

boolean       connect;
boolean       _debug = true;

void (*_apcallback)(WiFiManager*) = NULL;
void (*_savecallback)(void) = NULL;

WiFiManagerParameter* _params[WIFI_MANAGER_MAX_PARAMS];
```

```
    template <typename Generic>
    void          DEBUG_WM(Generic text);

    template <class T>
    auto optionalIPFromString(T *obj, const char *s) -> decltype(  obj->fromString(s)  ) {
      return  obj->fromString(s);
    }
    auto optionalIPFromString(...) -> bool {
      DEBUG_WM("NO fromString METHOD ON IPAddress, you need ESP8266 core 2.1.0 or newer for Custom IP configuration to work.");
      return false;
    }
};

#endif
```

```
//NOMBRE DEL ARCHIVO       WiFiManager.cpp


/**************************************************************
   WiFiManager is a library for the ESP8266/Arduino platform
   (https://github.com/esp8266/Arduino) to enable easy
   configuration and reconfiguration of WiFi credentials using a Captive Portal
   inspired by:
   http://www.esp8266.com/viewtopic.php?f=29&t=2520
   https://github.com/chriscook8/esp-arduino-apboot
   https://github.com/esp8266/Arduino/tree/master/libraries/DNSServer/examples/CaptivePortalAdvanced
   Built by AlexT https://github.com/tzapu
   Licensed under MIT license
 **************************************************************/

#include "WiFiManager.h"

WiFiManagerParameter::WiFiManagerParameter(const char *custom) {
  _id = NULL;
  _placeholder = NULL;
  _length = 0;
  _value = NULL;

  _customHTML = custom;
}

WiFiManagerParameter::WiFiManagerParameter(const char *id, const char *placeholder, const char *defaultValue, int length) {
  init(id, placeholder, defaultValue, length, "");
}

WiFiManagerParameter::WiFiManagerParameter(const char *id, const char *placeholder, const char *defaultValue, int length, const char
*custom) {
  init(id, placeholder, defaultValue, length, custom);
}

void WiFiManagerParameter::init(const char *id, const char *placeholder, const char *defaultValue, int length, const char *custom) {
  _id = id;
  _placeholder = placeholder;
  _length = length;
  _value = new char[length + 1];
  for (int i = 0; i < length; i++) {
    _value[i] = 0;
  }
  if (defaultValue != NULL) {
    strncpy(_value, defaultValue, length);
  }

  _customHTML = custom;
```

```
}

const char* WiFiManagerParameter::getValue() {
  return _value;
}
const char* WiFiManagerParameter::getID() {
  return _id;
}
const char* WiFiManagerParameter::getPlaceholder() {
  return _placeholder;
}
int WiFiManagerParameter::getValueLength() {
  return _length;
}
const char* WiFiManagerParameter::getCustomHTML() {
  return _customHTML;
}

WiFiManager::WiFiManager() {
}

void WiFiManager::addParameter(WiFiManagerParameter *p) {
  if(_paramsCount + 1 > WIFI_MANAGER_MAX_PARAMS)
  {
    //Max parameters exceeded!
    DEBUG_WM("WIFI_MANAGER_MAX_PARAMS exceeded, increase number (in WiFiManager.h) before adding more parameters!");
    DEBUG_WM("Skipping parameter with ID:");
    DEBUG_WM(p->getID());
    return;
  }
  _params[_paramsCount] = p;
  _paramsCount++;
  DEBUG_WM("Adding parameter");
  DEBUG_WM(p->getID());
}

void WiFiManager::setupConfigPortal() {
  dnsServer.reset(new DNSServer());
  server.reset(new ESP8266WebServer(80));

  DEBUG_WM(F(""));
  _configPortalStart = millis();

  DEBUG_WM(F("Configuring access point... "));
  DEBUG_WM(_apName);
  if (_apPassword != NULL) {
    if (strlen(_apPassword) < 8 || strlen(_apPassword) > 63) {
      // fail passphrase to short or long!
      DEBUG_WM(F("Invalid AccessPoint password. Ignoring"));
```

```
      _apPassword = NULL;
    }
    DEBUG_WM(_apPassword);
  }

  //optional soft ip config
  if (_ap_static_ip) {
    DEBUG_WM(F("Custom AP IP/GW/Subnet"));
    WiFi.softAPConfig(_ap_static_ip, _ap_static_gw, _ap_static_sn);
  }

  if (_apPassword != NULL) {
    WiFi.softAP(_apName, _apPassword);//password option
  } else {
    WiFi.softAP(_apName);
  }

  delay(500); // Without delay I've seen the IP address blank
  DEBUG_WM(F("AP IP address: "));
  DEBUG_WM(WiFi.softAPIP());

  /* Setup the DNS server redirecting all the domains to the apIP */
  dnsServer->setErrorReplyCode(DNSReplyCode::NoError);
  dnsServer->start(DNS_PORT, "*", WiFi.softAPIP());

  /* Setup web pages: root, wifi config pages, SO captive portal detectors and not found. */
  server->on("/", std::bind(&WiFiManager::handleRoot, this));
  server->on("/wifi", std::bind(&WiFiManager::handleWifi, this, true));
  server->on("/0wifi", std::bind(&WiFiManager::handleWifi, this, false));
  server->on("/wifisave", std::bind(&WiFiManager::handleWifiSave, this));
  server->on("/i", std::bind(&WiFiManager::handleInfo, this));
  server->on("/r", std::bind(&WiFiManager::handleReset, this));
  //server->on("/generate_204", std::bind(&WiFiManager::handle204, this));  //Android/Chrome OS captive portal check.
  server->on("/fwlink", std::bind(&WiFiManager::handleRoot, this));  //Microsoft captive portal. Maybe not needed. Might be handled
  by notFound handler.
  server->onNotFound (std::bind(&WiFiManager::handleNotFound, this));
  server->begin(); // Web server start
  DEBUG_WM(F("HTTP server started"));

}

boolean WiFiManager::autoConnect() {
  String ssid = "ESP" + String(ESP.getChipId());
  return autoConnect(ssid.c_str(), NULL);
}

boolean WiFiManager::autoConnect(char const *apName, char const *apPassword) {
  DEBUG_WM(F(""));
  DEBUG_WM(F("AutoConnect"));
```

```
  // read eeprom for ssid and pass
  //String ssid = getSSID();
  //String pass = getPassword();

  // attempt to connect; should it fail, fall back to AP
  WiFi.mode(WIFI_STA);

  if (connectWifi("", "") == WL_CONNECTED)   {
    DEBUG_WM(F("IP Address:"));
    DEBUG_WM(WiFi.localIP());
    //connected
    return true;
  }

  return startConfigPortal(apName, apPassword);
}

boolean WiFiManager::configPortalHasTimeout(){
    if(_configPortalTimeout == 0 || wifi_softap_get_station_num() > 0){
      _configPortalStart = millis(); // kludge, bump configportal start time to skew timeouts
      return false;
    }
    return (millis() > _configPortalStart + _configPortalTimeout);
}

boolean WiFiManager::startConfigPortal() {
  String ssid = "ESP" + String(ESP.getChipId());
  return startConfigPortal(ssid.c_str(), NULL);
}

boolean  WiFiManager::startConfigPortal(char const *apName, char const *apPassword) {
  //setup AP
  WiFi.mode(WIFI_AP_STA);
  DEBUG_WM("SET AP STA");

  _apName = apName;
  _apPassword = apPassword;

  //notify we entered AP mode
  if ( _apcallback != NULL) {
    _apcallback(this);
  }

  connect = false;
  setupConfigPortal();

  while(1){
```

```
    // check if timeout
    if(configPortalHasTimeout()) break;

    //DNS
    dnsServer->processNextRequest();
    //HTTP
    server->handleClient();


    if (connect) {
      connect = false;
      delay(2000);
      DEBUG_WM(F("Connecting to new AP"));

      // using user-provided  _ssid, _pass in place of system-stored ssid and pass
      if (connectWifi(_ssid, _pass) != WL_CONNECTED) {
        DEBUG_WM(F("Failed to connect."));
      } else {
        //connected
        WiFi.mode(WIFI_STA);
        //notify that configuration has changed and any optional parameters should be saved
        if ( _savecallback != NULL) {
          //todo: check if any custom parameters actually exist, and check if they really changed maybe
          _savecallback();
        }
        break;
      }

      if (_shouldBreakAfterConfig) {
        //flag set to exit after config after trying to connect
        //notify that configuration has changed and any optional parameters should be saved
        if ( _savecallback != NULL) {
          //todo: check if any custom parameters actually exist, and check if they really changed maybe
          _savecallback();
        }
        break;
      }
    }
    yield();
  }

  server.reset();
  dnsServer.reset();

  return  WiFi.status() == WL_CONNECTED;
}


int WiFiManager::connectWifi(String ssid, String pass) {
```

```
  DEBUG_WM(F("Connecting as wifi client..."));

  // check if we've got static_ip settings, if we do, use those.
  if (_sta_static_ip) {
    DEBUG_WM(F("Custom STA IP/GW/Subnet"));
    WiFi.config(_sta_static_ip, _sta_static_gw, _sta_static_sn);
    DEBUG_WM(WiFi.localIP());
  }
  //fix for auto connect racing issue
  if (WiFi.status() == WL_CONNECTED) {
    DEBUG_WM("Already connected. Bailing out.");
    return WL_CONNECTED;
  }
  //check if we have ssid and pass and force those, if not, try with last saved values
  if (ssid != "") {
    WiFi.begin(ssid.c_str(), pass.c_str());
  } else {
    if (WiFi.SSID()) {
      DEBUG_WM("Using last saved values, should be faster");
      //trying to fix connection in progress hanging
      ETS_UART_INTR_DISABLE();
      wifi_station_disconnect();
      ETS_UART_INTR_ENABLE();

      WiFi.begin();
    } else {
      DEBUG_WM("No saved credentials");
    }
  }

  int connRes = waitForConnectResult();
  DEBUG_WM ("Connection result: ");
  DEBUG_WM ( connRes );
  //not connected, WPS enabled, no pass - first attempt
  if (_tryWPS && connRes != WL_CONNECTED && pass == "") {
    startWPS();
    //should be connected at the end of WPS
    connRes = waitForConnectResult();
  }
  return connRes;
}

uint8_t WiFiManager::waitForConnectResult() {
  if (_connectTimeout == 0) {
    return WiFi.waitForConnectResult();
  } else {
    DEBUG_WM (F("Waiting for connection result with time out"));
    unsigned long start = millis();
    boolean keepConnecting = true;
```

```
    uint8_t status;
    while (keepConnecting) {
      status = WiFi.status();
      if (millis() > start + _connectTimeout) {
        keepConnecting = false;
        DEBUG_WM (F("Connection timed out"));
      }
      if (status == WL_CONNECTED || status == WL_CONNECT_FAILED) {
        keepConnecting = false;
      }
      delay(100);
    }
    return status;
  }
}

void WiFiManager::startWPS() {
  DEBUG_WM("START WPS");
  WiFi.beginWPSConfig();
  DEBUG_WM("END WPS");
}
/*
  String WiFiManager::getSSID() {
  if (_ssid == "") {
    DEBUG_WM(F("Reading SSID"));
    _ssid = WiFi.SSID();
    DEBUG_WM(F("SSID: "));
    DEBUG_WM(_ssid);
  }
  return _ssid;
  }

  String WiFiManager::getPassword() {
  if (_pass == "") {
    DEBUG_WM(F("Reading Password"));
    _pass = WiFi.psk();
    DEBUG_WM("Password: " + _pass);
    //DEBUG_WM(_pass);
  }
  return _pass;
  }
*/
String WiFiManager::getConfigPortalSSID() {
  return _apName;
}

void WiFiManager::resetSettings() {
  DEBUG_WM(F("settings invalidated"));
  DEBUG_WM(F("THIS MAY CAUSE AP NOT TO START UP PROPERLY. YOU NEED TO COMMENT IT OUT AFTER ERASING THE DATA."));
```

```
  WiFi.disconnect(true);
  //delay(200);
}
void WiFiManager::setTimeout(unsigned long seconds) {
  setConfigPortalTimeout(seconds);
}

void WiFiManager::setConfigPortalTimeout(unsigned long seconds) {
  _configPortalTimeout = seconds * 1000;
}

void WiFiManager::setConnectTimeout(unsigned long seconds) {
  _connectTimeout = seconds * 1000;
}

void WiFiManager::setDebugOutput(boolean debug) {
  _debug = debug;
}

void WiFiManager::setAPStaticIPConfig(IPAddress ip, IPAddress gw, IPAddress sn) {
  _ap_static_ip = ip;
  _ap_static_gw = gw;
  _ap_static_sn = sn;
}

void WiFiManager::setSTAStaticIPConfig(IPAddress ip, IPAddress gw, IPAddress sn) {
  _sta_static_ip = ip;
  _sta_static_gw = gw;
  _sta_static_sn = sn;
}

void WiFiManager::setMinimumSignalQuality(int quality) {
  _minimumQuality = quality;
}

void WiFiManager::setBreakAfterConfig(boolean shouldBreak) {
  _shouldBreakAfterConfig = shouldBreak;
}

/** Handle root or redirect to captive portal */
void WiFiManager::handleRoot() {
  DEBUG_WM(F("Handle root"));
  if (captivePortal()) { // If caprive portal redirect instead of displaying the page.
    return;
  }

  String page = FPSTR(HTTP_HEAD);
  page.replace("{v}", "Options");
  page += FPSTR(HTTP_SCRIPT);
```

```
  page += FPSTR(HTTP_STYLE);
  page += _customHeadElement;
  page += FPSTR(HTTP_HEAD_END);
  page += "<h1>";
  page += _apName;
  page += "</h1>";
  page += F("<h3>WiFiManager</h3>");
  page += FPSTR(HTTP_PORTAL_OPTIONS);
  page += FPSTR(HTTP_END);

  server->sendHeader("Content-Length", String(page.length()));
  server->send(200, "text/html", page);

}

/** Wifi config page handler */
void WiFiManager::handleWifi(boolean scan) {

  String page = FPSTR(HTTP_HEAD);
  page.replace("{v}", "Config ESP");
  page += FPSTR(HTTP_SCRIPT);
  page += FPSTR(HTTP_STYLE);
  page += _customHeadElement;
  page += FPSTR(HTTP_HEAD_END);

  if (scan) {
    int n = WiFi.scanNetworks();
    DEBUG_WM(F("Scan done"));
    if (n == 0) {
      DEBUG_WM(F("No networks found"));
      page += F("No networks found. Refresh to scan again.");
    } else {

      //sort networks
      int indices[n];
      for (int i = 0; i < n; i++) {
        indices[i] = i;
      }

      // RSSI SORT

      // old sort
      for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
          if (WiFi.RSSI(indices[j]) > WiFi.RSSI(indices[i])) {
            std::swap(indices[i], indices[j]);
          }
        }
      }
```

```
  /*std::sort(indices, indices + n, [](const int & a, const int & b) -> bool
    {
    return WiFi.RSSI(a) > WiFi.RSSI(b);
    });*/

  // remove duplicates ( must be RSSI sorted )
  if (_removeDuplicateAPs) {
    String cssid;
    for (int i = 0; i < n; i++) {
      if (indices[i] == -1) continue;
      cssid = WiFi.SSID(indices[i]);
      for (int j = i + 1; j < n; j++) {
        if (cssid == WiFi.SSID(indices[j])) {
          DEBUG_WM("DUP AP: " + WiFi.SSID(indices[j]));
          indices[j] = -1; // set dup aps to index -1
        }
      }
    }
  }

  //display networks in page
  for (int i = 0; i < n; i++) {
    if (indices[i] == -1) continue; // skip dups
    DEBUG_WM(WiFi.SSID(indices[i]));
    DEBUG_WM(WiFi.RSSI(indices[i]));
    int quality = getRSSIasQuality(WiFi.RSSI(indices[i]));

    if (_minimumQuality == -1 || _minimumQuality < quality) {
      String item = FPSTR(HTTP_ITEM);
      String rssiQ;
      rssiQ += quality;
      item.replace("{v}", WiFi.SSID(indices[i]));
      item.replace("{r}", rssiQ);
      if (WiFi.encryptionType(indices[i]) != ENC_TYPE_NONE) {
        item.replace("{i}", "l");
      } else {
        item.replace("{i}", "");
      }
      //DEBUG_WM(item);
      page += item;
      delay(0);
    } else {
      DEBUG_WM(F("Skipping due to quality"));
    }

  }
  page += "<br/>";
}
```

```
}

page += FPSTR(HTTP_FORM_START);
char parLength[5];
// add the extra parameters to the form
for (int i = 0; i < _paramsCount; i++) {
  if (_params[i] == NULL) {
    break;
  }

  String pitem = FPSTR(HTTP_FORM_PARAM);
  if (_params[i]->getID() != NULL) {
    pitem.replace("{i}", _params[i]->getID());
    pitem.replace("{n}", _params[i]->getID());
    pitem.replace("{p}", _params[i]->getPlaceholder());
    snprintf(parLength, 5, "%d", _params[i]->getValueLength());
    pitem.replace("{l}", parLength);
    pitem.replace("{v}", _params[i]->getValue());
    pitem.replace("{c}", _params[i]->getCustomHTML());
  } else {
    pitem = _params[i]->getCustomHTML();
  }

  page += pitem;
}
if (_params[0] != NULL) {
  page += "<br/>";
}

if (_sta_static_ip) {

  String item = FPSTR(HTTP_FORM_PARAM);
  item.replace("{i}", "ip");
  item.replace("{n}", "ip");
  item.replace("{p}", "Static IP");
  item.replace("{l}", "15");
  item.replace("{v}", _sta_static_ip.toString());

  page += item;

  item = FPSTR(HTTP_FORM_PARAM);
  item.replace("{i}", "gw");
  item.replace("{n}", "gw");
  item.replace("{p}", "Static Gateway");
  item.replace("{l}", "15");
  item.replace("{v}", _sta_static_gw.toString());

  page += item;
```

```
    item = FPSTR(HTTP_FORM_PARAM);
    item.replace("{i}", "sn");
    item.replace("{n}", "sn");
    item.replace("{p}", "Subnet");
    item.replace("{l}", "15");
    item.replace("{v}", _sta_static_sn.toString());

    page += item;

    page += "<br/>";
  }

  page += FPSTR(HTTP_FORM_END);
  page += FPSTR(HTTP_SCAN_LINK);

  page += FPSTR(HTTP_END);

  server->sendHeader("Content-Length", String(page.length()));
  server->send(200, "text/html", page);


  DEBUG_WM(F("Sent config page"));
}

/** Handle the WLAN save form and redirect to WLAN config page again */
void WiFiManager::handleWifiSave() {
  DEBUG_WM(F("WiFi save"));

  //SAVE/connect here
  _ssid = server->arg("s").c_str();
  _pass = server->arg("p").c_str();

  //parameters
  for (int i = 0; i < _paramsCount; i++) {
    if (_params[i] == NULL) {
      break;
    }
    //read parameter
    String value = server->arg(_params[i]->getID()).c_str();
    //store it in array
    value.toCharArray(_params[i]->_value, _params[i]->_length);
    DEBUG_WM(F("Parameter"));
    DEBUG_WM(_params[i]->getID());
    DEBUG_WM(value);
  }

  if (server->arg("ip") != "") {
    DEBUG_WM(F("static ip"));
    DEBUG_WM(server->arg("ip"));
```

```
    //_sta_static_ip.fromString(server->arg("ip"));
    String ip = server->arg("ip");
    optionalIPFromString(&_sta_static_ip, ip.c_str());
  }
  if (server->arg("gw") != "") {
    DEBUG_WM(F("static gateway"));
    DEBUG_WM(server->arg("gw"));
    String gw = server->arg("gw");
    optionalIPFromString(&_sta_static_gw, gw.c_str());
  }
  if (server->arg("sn") != "") {
    DEBUG_WM(F("static netmask"));
    DEBUG_WM(server->arg("sn"));
    String sn = server->arg("sn");
    optionalIPFromString(&_sta_static_sn, sn.c_str());
  }

  String page = FPSTR(HTTP_HEAD);
  page.replace("{v}", "Credentials Saved");
  page += FPSTR(HTTP_SCRIPT);
  page += FPSTR(HTTP_STYLE);
  page += _customHeadElement;
  page += FPSTR(HTTP_HEAD_END);
  page += FPSTR(HTTP_SAVED);
  page += FPSTR(HTTP_END);

  server->sendHeader("Content-Length", String(page.length()));
  server->send(200, "text/html", page);

  DEBUG_WM(F("Sent wifi save page"));

  connect = true; //signal ready to connect/reset
}

/** Handle the info page */
void WiFiManager::handleInfo() {
  DEBUG_WM(F("Info"));

  String page = FPSTR(HTTP_HEAD);
  page.replace("{v}", "Info");
  page += FPSTR(HTTP_SCRIPT);
  page += FPSTR(HTTP_STYLE);
  page += _customHeadElement;
  page += FPSTR(HTTP_HEAD_END);
  page += F("<dl>");
  page += F("<dt>Chip ID</dt><dd>");
  page += ESP.getChipId();
  page += F("</dd>");
  page += F("<dt>Flash Chip ID</dt><dd>");
```

```
  page += ESP.getFlashChipId();
  page += F("</dd>");
  page += F("<dt>IDE Flash Size</dt><dd>");
  page += ESP.getFlashChipSize();
  page += F(" bytes</dd>");
  page += F("<dt>Real Flash Size</dt><dd>");
  page += ESP.getFlashChipRealSize();
  page += F(" bytes</dd>");
  page += F("<dt>Soft AP IP</dt><dd>");
  page += WiFi.softAPIP().toString();
  page += F("</dd>");
  page += F("<dt>Soft AP MAC</dt><dd>");
  page += WiFi.softAPmacAddress();
  page += F("</dd>");
  page += F("<dt>Station MAC</dt><dd>");
  page += WiFi.macAddress();
  page += F("</dd>");
  page += F("</dl>");
  page += FPSTR(HTTP_END);

  server->sendHeader("Content-Length", String(page.length()));
  server->send(200, "text/html", page);

  DEBUG_WM(F("Sent info page"));
}

/** Handle the reset page */
void WiFiManager::handleReset() {
  DEBUG_WM(F("Reset"));

  String page = FPSTR(HTTP_HEAD);
  page.replace("{v}", "Info");
  page += FPSTR(HTTP_SCRIPT);
  page += FPSTR(HTTP_STYLE);
  page += _customHeadElement;
  page += FPSTR(HTTP_HEAD_END);
  page += F("Module will reset in a few seconds.");
  page += FPSTR(HTTP_END);

  server->sendHeader("Content-Length", String(page.length()));
  server->send(200, "text/html", page);

  DEBUG_WM(F("Sent reset page"));
  delay(5000);
  ESP.reset();
  delay(2000);
}

void WiFiManager::handleNotFound() {
```

```
  if (captivePortal()) { // If captive portal redirect instead of displaying the error page.
    return;
  }
  String message = "File Not Found\n\n";
  message += "URI: ";
  message += server->uri();
  message += "\nMethod: ";
  message += ( server->method() == HTTP_GET ) ? "GET" : "POST";
  message += "\nArguments: ";
  message += server->args();
  message += "\n";

  for ( uint8_t i = 0; i < server->args(); i++ ) {
    message += " " + server->argName ( i ) + ": " + server->arg ( i ) + "\n";
  }
  server->sendHeader("Cache-Control", "no-cache, no-store, must-revalidate");
  server->sendHeader("Pragma", "no-cache");
  server->sendHeader("Expires", "-1");
  server->sendHeader("Content-Length", String(message.length()));
  server->send ( 404, "text/plain", message );
}


/** Redirect to captive portal if we got a request for another domain. Return true in that case so the page handler do not try to
handle the request again. */
boolean WiFiManager::captivePortal() {
  if (!isIp(server->hostHeader()) ) {
    DEBUG_WM(F("Request redirected to captive portal"));
    server->sendHeader("Location", String("http://") + toStringIp(server->client().localIP()), true);
    server->send ( 302, "text/plain", ""); // Empty content inhibits Content-length header so we have to close the socket ourselves.
    server->client().stop(); // Stop is needed because we sent no content length
    return true;
  }
  return false;
}

//start up config portal callback
void WiFiManager::setAPCallback( void (*func)(WiFiManager* myWiFiManager) ) {
  _apcallback = func;
}

//start up save config callback
void WiFiManager::setSaveConfigCallback( void (*func)(void) ) {
  _savecallback = func;
}

//sets a custom element to add to head, like a new style tag
void WiFiManager::setCustomHeadElement(const char* element) {
  _customHeadElement = element;
```

```
}

//if this is true, remove duplicated Access Points - defaut true
void WiFiManager::setRemoveDuplicateAPs(boolean removeDuplicates) {
  _removeDuplicateAPs = removeDuplicates;
}



template <typename Generic>
void WiFiManager::DEBUG_WM(Generic text) {
  if (_debug) {
    Serial.print("*WM: ");
    Serial.println(text);
  }
}

int WiFiManager::getRSSIasQuality(int RSSI) {
  int quality = 0;

  if (RSSI <= -100) {
    quality = 0;
  } else if (RSSI >= -50) {
    quality = 100;
  } else {
    quality = 2 * (RSSI + 100);
  }
  return quality;
}

/** Is this an IP? */
boolean WiFiManager::isIp(String str) {
  for (int i = 0; i < str.length(); i++) {
    int c = str.charAt(i);
    if (c != '.' && (c < '0' || c > '9')) {
      return false;
    }
  }
  return true;
}

/** IP to String? */
String WiFiManager::toStringIp(IPAddress ip) {
  String res = "";
  for (int i = 0; i < 3; i++) {
    res += String((ip >> (8 * i)) & 0xFF) + ".";
  }
  res += String(((ip >> 8 * 3)) & 0xFF);
  return res;
```

```
}
```