```
/*----- Smart sunrise / sunset switch for the Eigthree ET28 with ESP32-C3 ------
For the Arduino IDE, select board to ESP32C3 dev module.
A brief hard- and software description is found at: https://www.larsenhenneberg.dk
-----
This output can set ON during day or night time using time values
for the sunrise/sunset time values.
An additional extra time be added via the webpage to extend the ON output time:

For the day time "extra time" option:
 |--------------|-------.-.-.-------|--------------|
   ½ extra time sunrise        sunset  ½ extra time

For the night time "extra time" option:
 |--------------|-------.-.-.-------|--------------|
   ½ extra time sunset        sunrise  ½ extra time
-----
Conditions: The program uses Arduino OTA to flash new firmware verions via wifi.
Make sure to use the correct values for:
Access point (AP) SSID and password to use the simple captive portal.
-----
Work scheme at reset or program restart:
.01: The wifi network ssid and password is read from the EEPROM
.02: The wifi connection is tried for a few sec. with the ssid and password, see .01
.03: In case of no wifi connection and and an active setup button, see .04
.04: Upon an active setup button, an Access Point (AP) is started, see .06
.05: The program continues into the loop(), see .11
.06: Use f.ex. a mobile phone to use the AP network.
.07: Browse to the captive portal at http://192.168.4.1
.08: Enter the wifi ssid and password at the captive portal web page and press Save.
.09: The ssid and password from the captive portal are saved into the EEprom.
.10: The ESP32 is restarted, and starts from .01
.11: In the loop(): in case of an active AP, the captive portal web client is handled.
.12: A while(1) loop inside loop() is now the main loop.
.13: The yield() function is called in while(1) to avoid a watch dog program reset.
.14: Upon an active setup button, the wifi is disconnected and the ESP32 is restarted, see .01
.15: If wifi is available, the NTP time is read on the hour, e.g. xx:00
.16: If no wifi is available, the time keeping is made with simple delay loops.
.17: If no wifi the output control uses the last saved Daylight Savings Time (DST)
-----
The sunrise/sunset config webpage is at address: http://ip_addr:http_port
or http://localURL.local:http_port . LocalURL is defined via a const char* in the code.

.Note1: The String htmlTitle defines the HTML page title
-----*/
#include <WiFi.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
#include <time.h>
#include <EEPROM.h>
#include <WebServer.h>

#define http_Port 8058        // http port to access the configuration webpage
#define TZ_DENMARK "CET-1CEST,M3.5.0,M10.5.0/3" // time zone and daylight def.

#define setup_Pin 5       // GPIO5 as active low input to invoke the captive portal
#define Blue_LED_pin 20   // GPIO20 as output , active low for blue LED network light
#define Red_LED_pin 21    // GPIO21 as output , active low for red LED network light
#define Relay1_pin 6      // GPIO6 as output , active high for Relay1
//#define SEL_PIN 4       // GPIO4 as output to select current or voltage for BL0937 CF1
```

```
//#define CF1_PIN 7       // GPIO7 as input to read current/voltage pulses, depending on SEL_PIN
#define CF_PIN 3          // GPIO6 as input to read BL0937 power metering pulses
#define ledON LOW
#define ledOFF HIGH
#define relayON HIGH
#define relayOFF LOW

//for power monitoring
#define POWER_MULTIPLIER 1.47 // factor to multiply to the CF pulse to get the power in W
#define CFtimeout 5000        // power calc. timeout in msec. after last CF ISR interrupt

// Allocate 512 bytes for wifi ssid, passw. and daynight boolean and added time to the ON period
#define EEPROM_SIZE 512
#define EEprom_adr 0        // EEprom start address normally 0

String htmlTitle = "Smart sunrise/sunset switch";  // web page title
const char* localURL = "sunrise-sunset_timer";     // use sunrise-sunset_timer.local as LAN URL

// Soft AP ( Access Point) network setup for the captive portal
const char* APssid = "ESP32-C3 sunrise-set switch";
const char* APpassword = "";              // might add AP password here !

WebServer APserver(80);       // this webservice is for AP only
WiFiServer server(http_Port); // webservice for the timeplan setup

bool daynight;    //  0=daytime ON, 1=nighttime ON, savec in EEprom

bool ntpTrig = true;      // to enable NTP request at first loop entry

bool outputState = false; // output relay state indicator

uint16_t addONminutes;  // # minutes to add to the relay ON period, saved in EEprom
uint16_t addONminutesMax = 60;    // max #minutes to add to relay ON period
uint16_t DST_offset_minutes;        // DK DST offset 60 or 120 min.
uint16_t sunriseDKminutes, sunsetDKminutes;

// values in msec used by millis() for non-blocking loop() processing
uint32_t previousMillis = 0;
uint32_t OneMinute = 60 * 1000L;

// power metering variables
float powerW;
float pulseFreq;
uint32_t pulseWidth, msNow, msPreviousPulse;
void ICACHE_RAM_ATTR CF_impulse();

// globals for Network Time Protocol ( NTP ) response
struct tm timeinfo;
time_t now;

// globals to hold the danish NTP variables
String Hour_str;
String Minute_str;
String Month_str;
String Day_str;
String DayOfMonth_str;
String Year_str;
String timestr;

byte DayOfMonth;  // Day of month 0..30
```

```
byte Month;       // Month 1..12
byte Hour;        // Hours 0..23
byte Minute;      // Minutes 0..59
uint16_t Year;        // Year 2024..

// globals to hold the NTP UTC hour
String UTC_timestr;
String UTC_Hour_str;
byte UTC_Hour;        // Hours 0..23

// number of days per month for leap and non-leap years
const byte daysPerMonth_Leap[12] = {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
const byte daysPerMonth_Normal[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

// EEprom contents for wifi are read in at start, and updated on changes.
struct wifiSettings {   // the wifiSettings
  char ssid[30];
  char password[30];
} user_wifi = {};

// dataset for the hardcoded sunrise/sunset arrays
typedef struct
{
  uint16_t sunrise; // holds the number of minutes from 00:00 to sunrise
  uint16_t sunset;  // holds the number of minutes from 00:00 to sunset
} rise_set;

// The sunrise_sun array originate from https://aa.usno.navy.mil/data/RS_OneYear for the year
2024, used for the leap years
// Each month got space for 31 days , but of course just 7 months a year utilizes all 31 entries
for the days of a month.!
// The array entries are UTC with location for Sønderborg Denmark, that is: longitude=54,9183 and
latitude=9,8200
// The sunrise, sunset datasets are the number of minutes (60*hour+minutes) from 00:00 for each
day of the year
const rise_set sunrise_sunset_leap[12][31] = {
/* jan */ {{465,903},{465,904},{465,905},{464,907},{464,908},{463,909},{463,911},{462,912},{462,914
},{461,915},{460,917},{459,919},{459,920},{458,922},{457,924},{456,926},{454,927},{453,929},{452,
931},{451,933},{449,935},{448,937},{447,939},{445,941},{444,943},{442,945},{441,947},{439,949},{437
,951},{436,953},{434,955}},
/* feb */ {{432,957},{430,959},{429,961},{427,963},{425,965},{423,967},{421,969},{419,972},{417,974
},{415,976},{413,978},{411,980},{409,982},{406,984},{404,986},{402,988},{400,990},{398,993},{395,
995},{393,997},{391,999},{389,1001},{386,1003},{384,1005},{382,1007},{379,1009},{377,1011},{374,
1013},{372,1015},{0,0},{0,0}},
/* mar */ {{370,1017},{368,1019},{365,1021},{362,1023},{360,1025},{357,1027},{355,1029},{352,1031
},{350,1033},{347,1035},{345,1037},{342,1039},{340,1041},{337,1043},{335,1045},{332,1047},{330,1049
},{327,1051},{325,1053},{322,1055},{320,1057},{317,1059},{314,1061},{312,1063},{309,1065},{307,1067
},{304,1069},{302,1071},{299,1073},{297,1075},{294,1077}},
/* apr */ {{292,1078},{289,1080},{287,1082},{284,1084},{282,1086},{279,1088},{277,1090},{274,1092
},{272,1094},{269,1096},{267,1098},{264,1100},{262,1102},{259,1104},{257,1106},{254,1108},{252,1110
},{250,1112},{247,1113},{245,1115},{243,1117},{240,1119},{238,1121},{236,1123},{233,1125},{231,1127
},{229,1129},{227,1131},{225,1133},{222,1135},{0,0}},
/* may */ {{220,1137},{218,1139},{216,1140},{214,1142},{212,1144},{210,1146},{208,1148},{206,1150
},{204,1152},{202,1153},{200,1155},{198,1157},{196,1159},{195,1161},{193,1162},{191,1164},{189,1166
},{188,1168},{186,1169},{185,1171},{183,1173},{182,1174},{180,1176},{179,1177},{178,1179},{176,1180
},{175,1182},{174,1183},{173,1185},{172,1186},{171,1187}},
/* jun */ {{170,1189},{169,1190},{168,1191},{167,1192},{166,1193},{165,1194},{165,1195},{164,1196
},{164,1197},{163,1198},{163,1199},{162,1199},{162,1200},{162,1201},{162,1201},{162,1202},{162,1202
},{162,1202},{162,1203},{162,1203},{162,1203},{162,1203},{163,1203},{163,1203},{163,1203},{164,1203
},{164,1203},{165,1203},{166,1203},{166,1202},{0,0}},
```

```
/* jul */ {{167,1202},{168,1201},{169,1201},{170,1200},{171,1199},{172,1199},{173,1198},{174,1197
},{175,1196},{176,1195},{178,1194},{179,1193},{180,1192},{182,1191},{183,1190},{184,1188},{186,1187
},{187,1186},{189,1184},{190,1183},{192,1181},{194,1180},{195,1178},{197,1177},{199,1175},{200,1173
},{202,1171},{204,1170},{205,1168},{207,1166},{209,1164}},
/* aug */ {{211,1162},{212,1160},{214,1158},{216,1156},{218,1154},{220,1152},{221,1150},{223,1148
},{225,1146},{227,1144},{229,1142},{231,1139},{233,1137},{234,1135},{236,1133},{238,1130},{240,1128
},{242,1126},{244,1124},{246,1121},{247,1119},{249,1116},{251,1114},{253,1112},{255,1109},{257,1107
},{259,1104},{260,1102},{262,1099},{264,1097},{266,1094}},
/* sep */ {{268,1092},{270,1089},{272,1087},{273,1084},{275,1082},{277,1079},{279,1077},{281,1074
},{283,1072},{285,1069},{286,1067},{288,1064},{290,1062},{292,1059},{294,1056},{296,1054},{298,1051
},{299,1049},{301,1046},{303,1044},{305,1041},{307,1039},{309,1036},{311,1033},{312,1031},{314,1028
},{316,1026},{318,1023},{320,1021},{322,1018},{0,0}},
/* oct */ {{324,1016},{326,1013},{328,1011},{330,1008},{331,1006},{333,1003},{335,1001},{337,998},{
339,996},{341,993},{343,991},{345,988},{347,986},{349,983},{351,981},{353,979},{355,976},{357,974
},{359,971},{361,969},{363,967},{365,965},{367,962},{369,960},{371,958},{373,956},{375,953},{377,
951},{379,949},{381,947},{383,945}},
/* nov */ {{385,943},{387,941},{389,939},{391,937},{393,935},{395,933},{397,931},{399,929},{401,927
},{403,925},{405,924},{407,922},{409,920},{411,919},{413,917},{415,915},{417,914},{419,912},{421,
911},{423,910},{425,908},{426,907},{428,906},{430,904},{432,903},{434,902},{435,901},{437,900},{439
,899},{440,898},{0,0}},
/* dec */ {{442,898},{443,897},{445,896},{446,895},{448,895},{449,894},{451,894},{452,894},{453,893
},{454,893},{455,893},{456,893},{457,893},{458,893},{459,893},{460,893},{461,893},{462,893},{462,
894},{463,894},{463,895},{464,895},{464,896},{465,896},{465,897},{465,898},{465,899},{465,900},{465
,901},{465,902},{465,903}}
};

// The sunrise_sun array originate from https://aa.usno.navy.mil/data/RS_OneYear for the year
2023, used for the non-leap years
// Each month got space for 31 days , but of course just 7 months a year utilizes all 31 entries
for the days of a month.!
// The array entries are UTC with location for Sønderborg Denmark, that is: longitude=54,9183 and
latitude=9,8200
// The sunrise, sunset datasets are the number of minutes (60*hour+minutes) from 00:00 for each
day of the year
const rise_set sunrise_sunset_normal[12][31] = {
/* jan */ {{465,903},{465,905},{465,906},{464,907},{464,908},{463,910},{463,911},{462,913},{462,914
},{461,916},{460,917},{459,919},{458,921},{457,922},{456,924},{455,926},{454,928},{453,930},{452,
931},{450,933},{449,935},{448,937},{446,939},{445,941},{443,943},{442,945},{440,947},{439,949},{437
,951},{435,953},{434,955}},
/* feb */ {{432,957},{430,959},{428,962},{426,964},{424,966},{422,968},{420,970},{418,972},{416,974
},{414,976},{412,978},{410,980},{408,983},{406,985},{404,987},{402,989},{399,991},{397,993},{395,
995},{393,997},{390,999},{388,1001},{386,1003},{383,1006},{381,1008},{379,1010},{376,1012},{374,
1014},{0,0},{0,0},{0,0}},
/* mar */ {{371,1016},{369,1018},{367,1020},{364,1022},{362,1024},{359,1026},{357,1028},{354,1030
},{352,1032},{349,1034},{347,1036},{344,1038},{342,1040},{339,1042},{337,1044},{334,1046},{332,1048
},{329,1050},{327,1052},{324,1054},{321,1056},{319,1058},{316,1059},{314,1061},{311,1063},{309,1065
},{306,1067},{304,1079},{301,1071},{299,1073},{296,1075}},
/* apr */ {{294,1077},{291,1079},{289,1081},{286,1083},{283,1085},{281,1087},{278,1089},{276,1091
},{273,1092},{271,1094},{269,1096},{266,1098},{264,1100},{261,1102},{259,1104},{256,1106},{254,1108
},{251,1110},{249,1112},{247,1114},{244,1116},{242,1118},{240,1120},{237,1122},{235,1124},{233,1126
},{231,1128},{228,1129},{226,1131},{224,1133},{0,0}},
/* may */ {{222,1135},{220,1137},{218,1139},{215,1141},{213,1143},{211,1145},{209,1147},{207,1148
},{205,1150},{203,1152},{201,1154},{200,1156},{198,1158},{196,1159},{194,1161},{192,1163},{191,1165
},{189,1166},{187,1168},{186,1170},{184,1171},{183,1173},{181,1175},{180,1176},{179,1178},{177,1179
},{176,1181},{175,1182},{174,1184},{172,1185},{171,1186}},
/* jun */ {{170,1188},{169,1189},{168,1190},{168,1191},{167,1192},{166,1193},{165,1194},{165,1195
},{164,1196},{164,1197},{163,1198},{163,1199},{162,1200},{162,1200},{162,1201},{162,1201},{162,1202
},{162,1202},{162,1203},{162,1203},{162,1203},{162,1203},{162,1203},{163,1203},{163,1203},{164,1203
},{164,1203},{165,1203},{165,1202},{166,1202},{0,0}},
/* jul */ {{167,1202},{168,1202},{168,1201},{169,1201},{170,1200},{171,1199},{172,1198},{173,1198
```

```
},{174,1197},{176,1196},{177,1195},{178,1194},{179,1193},{181,1192},{182,1191},{183,1189},{185,1188
},{186,1187},{188,1185},{189,1184},{191,1182},{192,1181},{194,1179},{196,1178},{197,1176},{199,1174
},{201,1173},{202,1171},{204,1169},{206,1167},{208,1166}},
/* aug */ {{209,1164},{211,1162},{213,1160},{215,1158},{216,1156},{218,1154},{220,1152},{222,1150
},{224,1148},{226,1145},{227,1143},{229,1141},{231,1139},{233,1137},{235,1134},{237,1132},{239,1130
},{240,1128},{242,1125},{244,1123},{246,1121},{248,1118},{250,1116},{252,1113},{253,1111},{255,1109
},{257,1106},{259,1104},{261,1101},{263,1099},{265,1096}},
/* sep */ {{266,1094},{268,1091},{270,1089},{272,1086},{274,1084},{276,1081},{278,1079},{279,1076
},{281,1074},{283,1071},{285,1069},{287,1066},{289,1064},{291,1061},{292,1058},{294,1056},{296,1053
},{298,1051},{300,1049},{302,1046},{304,1043},{305,1040},{307,1038},{309,1035},{311,1033},{313,1030
},{315,1028},{317,1025},{319,1023},{320,1020},{0,0}},
/* oct */ {{322,1018},{324,1016},{326,1013},{328,1011},{330,1008},{331,1006},{333,1003},{335,1001
},{337,998},{339,996},{341,993},{343,991},{345,988},{347,986},{349,983},{351,981},{353,979},{355,
976},{357,974},{359,971},{361,969},{363,967},{365,965},{367,962},{369,960},{371,958},{373,956},{375
,953},{377,951},{379,949},{381,947}},
/* nov */ {{383,944},{385,942},{387,940},{390,938},{392,936},{394,934},{396,932},{398,931},{400,929
},{402,927},{404,925},{406,923},{408,922},{410,920},{412,918},{414,917},{416,915},{417,914},{419,
912},{421,911},{423,909},{425,908},{427,907},{429,905},{430,904},{432,903},{434,902},{436,901},{437
,900},{439,899},{0,0}},
/* dec */ {{441,898},{442,897},{444,897},{445,896},{447,895},{448,895},{450,894},{451,894},{452,893
},{453,893},{455,893},{456,893},{457,893},{458,893},{459,893},{460,893},{460,893},{461,893},{462,
893},{463,894},{463,894},{464,895},{464,895},{464,896},{465,897},{465,897},{465,898},{465,899},{465
,900},{465,901},{465,902}}
};

void setup() {
  pinMode(setup_Pin, INPUT_PULLUP);
  pinMode(Blue_LED_pin, OUTPUT);   // Blue LED pin as output
  pinMode(Red_LED_pin, OUTPUT);    // Red LED pin as output
  pinMode(Relay1_pin, OUTPUT);     // Relay1 pin as output
  pinMode(CF_PIN, INPUT);             // Set pin to input for capturing CF_PIN pulses
  digitalWrite(Blue_LED_pin, ledOFF); // set blue LED inactive
  digitalWrite(Red_LED_pin, ledOFF);  // set red LED inactive
  digitalWrite(Relay1_pin, relayOFF);  // set Relay1 output inactive
  interrupts();                                  // Enable interrupts (in case they were previously
disabled)
  attachInterrupt(digitalPinToInterrupt(CF_PIN), CF_impulse, RISING); // power pulse interrupt

  EEPROM.begin(EEPROM_SIZE);
  EEPROM.get(0, user_wifi);
  EEPROM.get(sizeof(struct wifiSettings), addONminutes);
  EEPROM.get(sizeof(struct wifiSettings) + sizeof(addONminutes), daynight);
  // to avoid large an addONminutes value read from an uninitialized EEprom:
  if ( addONminutes > addONminutesMax ) { addONminutes = addONminutesMax; }
// blink LED while trying the wifi connection with ssid,password from EEprom
// stay here while no wifi connection AND no setup button being pressed
  WiFi.mode(WIFI_STA);
  WiFi.begin(user_wifi.ssid, user_wifi.password);
  while ((WiFi.status() != WL_CONNECTED) && (!setupButton())) {
    digitalWrite(Blue_LED_pin, 0);
    delay(500);
    digitalWrite(Blue_LED_pin, 1);
    delay(500);
  }
  // on no wifi OR setup button pressed: enter the captive portal
  if ((WiFi.status() != WL_CONNECTED) || (setupButton())) {
    WiFi.mode(WIFI_AP);
    WiFi.softAP(APssid, APpassword);
    APserver.on("/",  handlePortal);      // start captive portal
    APserver.begin();
```

```
  }
  // start the "Over The Air", OTA firmware update option
  initOTA();
  server.begin();

  // set the local URL
  int cnt = 5;
  while (!MDNS.begin(localURL) && (cnt > 0)) {
    delay(1000);
    cnt--;
  }
  previousMillis = millis(); // init internal time keeping counter
  msPreviousPulse = millis();
}

void loop() {
  // on no wifi, short blink LED and handle the captive portal
  while (WiFi.status() != WL_CONNECTED) { // waiting for AP client
    digitalWrite(Blue_LED_pin, 0);
    delay(100);                       // short blink to indicate
    digitalWrite(Blue_LED_pin, 1);    // an active captive portal
    delay(500);
    APserver.handleClient();
  }
  APserver.close();   // Close the APserver for the captive portal
  while (1) {             // going into the main loop !
    yield();              // refresh wdt (watch dog timer) to avoid wdt reset
    if (setupButton()) {  // setup button pressed ?
      WiFi.disconnect();  // disconnect wifi
      ESP.restart();      // restart to continue with AP
    }

  if ((millis() - msPreviousPulse) > CFtimeout ) { powerW = 0.0; }
  else { powerW = pulseFreq * POWER_MULTIPLIER + 0.5; } // CF frequency to power in W, rounded up

  // if wifi available then request NTP regularly
  if ((WiFi.status() == WL_CONNECTED) && (ntpTrig == true)) {
    digitalWrite(Blue_LED_pin, 0); // set blue LED active
    configTime(0, 0, "3.dk.pool.ntp.org");    // First connect to NTP server, with 0 TZ offset
    if (getLocalTime(&timeinfo)) {
      char timeStringBuff[30];      // to hold timeinfo characters
      strftime(timeStringBuff, sizeof(timeStringBuff), "%a %b %d %H:%M:%S %Y", &timeinfo);
      UTC_timestr = String(timeStringBuff);  // ex.: Wed Dec 27 17:43:24 2023
      UTC_Hour_str = UTC_timestr.substring(11,13);
      UTC_Hour = UTC_Hour_str.toInt();         // save hour as byte
      setenv("TZ",TZ_DENMARK,1);  // Adjust to the TZ and Daylight Savings Time (DST).
      tzset();                    // Clock settings are adjusted to show the new local time
      if (getLocalTime(&timeinfo)) {
        char timeStringBuff[30];      // to hold timeinfo characters
        strftime(timeStringBuff, sizeof(timeStringBuff), "%a %b %d %H:%M:%S %Y", &timeinfo);
        timestr = String(timeStringBuff);  // ex.: Wed Dec 27 17:43:24 2023
        Day_str = timestr.substring(0,3);
        Month_str = timestr.substring(4,7);
        Month = 0;  // month index 0..11 for sunrise/sunset lookup
        if (Month_str == "Jan") { Month = 0; }
        if (Month_str == "Feb") { Month = 1; }
        if (Month_str == "Mar") { Month = 2; }
        if (Month_str == "Apr") { Month = 3; }
        if (Month_str == "May") { Month = 4; }
        if (Month_str == "Jun") { Month = 5; }
```

```
      if (Month_str == "Jul") { Month = 6; }
      if (Month_str == "Aug") { Month = 7; }
      if (Month_str == "Sep") { Month = 8; }
      if (Month_str == "Oct") { Month = 9; }
      if (Month_str == "Nov") { Month = 10; }
      if (Month_str == "Dec") { Month = 11; }
      DayOfMonth_str = timestr.substring(8,10);
      DayOfMonth = DayOfMonth_str.toInt() - 1; // Month index 0..30
      Hour_str = timestr.substring(11,13);
      Hour = Hour_str.toInt();         // save hour as byte
      DST_offset_minutes = (Hour - UTC_Hour) * 60;  // DK DST offset 60 or 120 min.
      Minute_str = timestr.substring(14,16);
      Minute = Minute_str.toInt();     // save minute as byte
      Year_str = timestr.substring(20,24);
      Year = Year_str.toInt();
      ntpTrig = false;
    }  // if gettime DK
  } // if gettime UTC
}   // wifi and getNtp
if (millis() - previousMillis >= OneMinute) {
  previousMillis = millis();
  Minute++;
  if (Minute >= 60) {
    ntpTrig = true;       // get NTP on each hour
    Minute = 0;
    Hour++;
    if (Hour >= 24) { // passed midnight ?
      byte lastDayOfMonth;
      Hour = 0;
      DayOfMonth++;
      if ((Year % 4) == 0) {  // leap year ?
        lastDayOfMonth = daysPerMonth_Leap[Month] - 1;
      }
      else {
        lastDayOfMonth = daysPerMonth_Normal[Month] - 1;
      }
      if (DayOfMonth > lastDayOfMonth) { // next month ?
        DayOfMonth = 0;   // first day , next month
        Month++;
        if (Month > 11) {  // past newyear ?
          Month = 0;
          Year++;
        }
      }
    }
  }
} // millis()
updateOutput();
// if wifi: look for an OTA update and client request
if ((WiFi.status() == WL_CONNECTED)) {
 digitalWrite(Blue_LED_pin, 0); // set blue LED active
 ArduinoOTA.handle();
 WiFiClient client = server.available();
 if (client) {
  String currentLine = "";                // make a String to hold incoming data from the client
  String header = "";                     // Variable to store the HTTP request
  while (client.connected()) { // loop while the client's connected
    if (client.available()) {             // if there's bytes to read from the client,
      char c = client.read();             // read a byte, then
      header += c;                        // add to the header
```

```
        if (c == '\n') {                    // if end of the client HTTP request
          if (currentLine.length() == 0) {  // then send a response:
            client.println("HTTP/1.0 200 OK");    // send HTTP header
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();
            if (header.indexOf("extmins") >= 0) { // look for "extmins" in URL
              int idx = header.indexOf("extmins");
              char minMSB = header[idx+8] - '0';
              char minLSB = header[idx+9] - '0';
              uint16_t ext_period = minMSB*10 + minLSB;
              if (addONminutes != ext_period) {   // EEprom update if new <> old value
                addONminutes = ext_period;
                EEPROM.put(sizeof(struct wifiSettings), addONminutes);
                EEPROM.commit();
              }
            }
            if (header.indexOf("daynight") >= 0) { // look for "daynight" in URL
              int idx = header.indexOf("daynight");
              if (header[idx+9] == '0') {
                if (daynight != false) { // EEprom update if new <> old value
                  daynight = false;
                  EEPROM.put(sizeof(struct wifiSettings) + sizeof(addONminutes), daynight);
                  EEPROM.commit();
                }
              }
              else {  // header[idx+9] == '1'
                if (daynight != true) {  // EEprom update if new <> old value
                  daynight = true;
                  EEPROM.put(sizeof(struct wifiSettings) + sizeof(addONminutes), daynight);
                  EEPROM.commit();
                }
              }
            }
            updateOutput();  // update output state before building client html reponse
            int P10 = int((powerW + 0.5) * 10);
            String s, pwrStr, outputState_str = "The state of the output: ";
            String extra2Rows1col = "<tr><td colspan=\"1\"></td></tr><tr><td
colspan=\"1\"></td></tr>";
            String tdStyle = "<style>td {  font-weight:bold; text-align: left; }</style>";
            pwrStr = "power consumption: " + String(int(powerW*10) / 10) + "." + String(int(powerW*
10) % 10) + " W";
            if (outputState == true) { outputState_str += "ON, "; }
            else { outputState_str += "OFF, "; }
            client.println("<!DOCTYPE html><html lang=\"en\">");
            s = "<head><title>" + htmlTitle + "</title><meta charset=\"utf-8\">";
//            s += "<meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\">";
            s += tdStyle + "</head><body>";
            client.println(s);
            client.println("<table style=\"width:100%\"><tr><td>Smart plug switching based on
sunrise/sunset</td></tr>" + extra2Rows1col);
            s = "<tr><td>" + outputState_str + pwrStr + "</td></tr>" + extra2Rows1col;
            s += "<tr><td>" + Day_str + " " + DayOfMonth_str + ". " + Month_str + " " + " " +
Year_str + " " + time2String(Hour, Minute);
            if ((Year % 4) == 0) {  // leap year ?
              s += ", sunrise: " + time2String((sunrise_sunset_leap[Month][DayOfMonth].sunrise +
DST_offset_minutes)/60,(sunrise_sunset_leap[Month][DayOfMonth].sunrise + DST_offset_minutes)%60);
              s += ", sunset: " + time2String((sunrise_sunset_leap[Month][DayOfMonth].sunset +
DST_offset_minutes)/60,(sunrise_sunset_leap[Month][DayOfMonth].sunset + DST_offset_minutes)%60)  +
"</td></tr></table>";
```

```
            }
            else {
              s += ", sunrise: " + time2String((sunrise_sunset_normal[Month][DayOfMonth].sunrise +
DST_offset_minutes)/60,(sunrise_sunset_normal[Month][DayOfMonth].sunrise + DST_offset_minutes)%60);
              s += ", sunset: " + time2String((sunrise_sunset_normal[Month][DayOfMonth].sunset +
DST_offset_minutes)/60,(sunrise_sunset_normal[Month][DayOfMonth].sunset + DST_offset_minutes)%60)
+ "</td></tr></table>";
            }
            s += "<form><table><tr><td colspan=\"2\"></td></tr><tr><td>Active ON period:</td><td>";
            s += "<input type=\"radio\" name=\"daynight\" onclick=\"this.form.submit()\"
value=\"0\"";
            if (daynight == false) { s += " checked"; }
            s += ">daytime<input type=\"radio\" name=\"daynight\" onclick=\"this.form.submit()\"
value=\"1\"";
            if (daynight == true) { s += " checked"; }
            s += ">nighttime</td></tr></table></form>";
            if ((Year % 4) == 0) {  // leap year ?
              sunriseDKminutes = sunrise_sunset_leap[Month][DayOfMonth].sunrise +
DST_offset_minutes - addONminutes/2;
              sunsetDKminutes = sunrise_sunset_leap[Month][DayOfMonth].sunset + DST_offset_minutes
+ addONminutes/2;
            }
            else {
              sunriseDKminutes = sunrise_sunset_normal[Month][DayOfMonth].sunrise +
DST_offset_minutes - addONminutes/2;
              sunsetDKminutes = sunrise_sunset_normal[Month][DayOfMonth].sunset +
DST_offset_minutes + addONminutes/2;
            }
            byte onHour = sunriseDKminutes/60;
            byte onMin = sunriseDKminutes % 60;
            byte offHour = sunsetDKminutes/60;
            byte offMin = sunsetDKminutes % 60;
            s += "<table><tr><td colspan=\"1\"></td></tr><tr><td>";
            if (daynight == 0) {
              s += "Daytime period switch ON, OFF times: " + time2String(onHour, onMin) + ", " +
time2String(offHour, offMin);
            }
            else {
              if ((Year % 4) == 0) {  // leap year ?
                      sunriseDKminutes = sunrise_sunset_leap[Month][DayOfMonth].sunrise +
DST_offset_minutes + addONminutes/2;
                    sunsetDKminutes = sunrise_sunset_leap[Month][DayOfMonth].sunset +
DST_offset_minutes - addONminutes/2;
              }
              else {
                      sunriseDKminutes = sunrise_sunset_normal[Month][DayOfMonth].sunrise +
DST_offset_minutes + addONminutes/2;
                    sunsetDKminutes = sunrise_sunset_normal[Month][DayOfMonth].sunset +
DST_offset_minutes - addONminutes/2;
              }
                    onHour = sunriseDKminutes/60;
                    onMin = sunriseDKminutes % 60;
                    offHour = sunsetDKminutes/60;
                    offMin = sunsetDKminutes % 60;
              s += "Nighttime period switch ON, OFF times: " + time2String(offHour, offMin) + ", "
+ time2String(onHour, onMin);
            }
            s += "</td></tr>" + extra2Rows1col;
            s += "<tr><td>Currently added ON time: " + String(addONminutes) + " min.</td></tr>" +
extra2Rows1col + "</table>";
```

```
            s += "<table><tr><td>Select added ON time: </td>";
            s += "<td><form><select name=\"extmins\" onchange=\"this.form.submit()\">";
            s += "<option value=\"\">Add ON time</option>";
            s += "<option value=\"00\">0 min.</option>";
            s += "<option value=\"15\">15 min.</option>";
            s += "<option value=\"30\">30 min.</option>";
            s += "<option value=\"45\">45 min.</option>";
            s += "<option value=\"60\">60 min.</option>";
            s += "</select></form></td></tr></table></body></html>";
            client.println(s);
            client.println();
            break;
          } // len == 0
          else { // if you got a newline, then clear currentLine
            currentLine = "";
          }
        } // if c == '\n'
        else {
          if (c != '\r') {  // if you got anything else but a carriage return character,
            currentLine += c;      // add it to the end of the currentLine
          }
        }
      } // if client available
      yield();
    }  //  while client connected
    header = "";
    client.stop();
   } // if client
  }  // if wifi connected
  else {  // if no wifi: set blue LED inactive
   digitalWrite(Blue_LED_pin, 1);
  }
  }  //  while(1)
} // loop()

// check the DK time against the sunrise/sunset values, taking DST offset and added ON time into
account
void updateOutput() {
  uint16_t timeNow_minutes = Hour*60 + Minute;
  if (daynight == false) {
    if ((Year % 4) == 0) {  // leap year ?
      sunriseDKminutes = sunrise_sunset_leap[Month][DayOfMonth].sunrise + DST_offset_minutes -
addONminutes/2;
      sunsetDKminutes = sunrise_sunset_leap[Month][DayOfMonth].sunset + DST_offset_minutes +
addONminutes/2;
    }
    else {
      sunriseDKminutes = sunrise_sunset_normal[Month][DayOfMonth].sunrise + DST_offset_minutes -
addONminutes/2;
      sunsetDKminutes = sunrise_sunset_normal[Month][DayOfMonth].sunset + DST_offset_minutes +
addONminutes/2;
    }
    if ((timeNow_minutes >= sunriseDKminutes) && (timeNow_minutes <= sunsetDKminutes)) {
      outputState = true;
    }
    else { outputState = false; }
  }
  else {
    if ((Year % 4) == 0) {  // leap year ?
      sunriseDKminutes = sunrise_sunset_leap[Month][DayOfMonth].sunrise + DST_offset_minutes +
```

```
addONminutes/2;
      sunsetDKminutes = sunrise_sunset_leap[Month][DayOfMonth].sunset + DST_offset_minutes -
addONminutes/2;
    }
    else {
      sunriseDKminutes = sunrise_sunset_normal[Month][DayOfMonth].sunrise + DST_offset_minutes +
addONminutes/2;
      sunsetDKminutes = sunrise_sunset_normal[Month][DayOfMonth].sunset + DST_offset_minutes -
addONminutes/2;
    }
    if ((timeNow_minutes >= sunsetDKminutes) || (timeNow_minutes <= sunriseDKminutes)) {
      outputState = true;
    }
    else { outputState = false; }
  }
  // set output relay pin and red LED pin according to outputState
  if (outputState == true) {
    digitalWrite(Relay1_pin, 1);    // set Relay1 output active
    digitalWrite(Red_LED_pin, 0);   // set red LED active
  }
  else {
    digitalWrite(Relay1_pin, 0);    // disable Relay1 output
    digitalWrite(Red_LED_pin, 1);   // set red LED inactive
  }
}

String time2String(byte hour, byte min) {
  String time_2digits = "";
  if (hour <= 9) { time_2digits += "0"; }
  time_2digits += String(hour) + ":";
  if (min <= 9) { time_2digits += "0"; }
  time_2digits += String(min);
  return time_2digits;
}

bool setupButton() {
  bool result = false;
  if (digitalRead(setup_Pin) == 0) {
    delay(20);    // simple anti bounce delay
    if (digitalRead(setup_Pin) == 0) { result = true; }
  }
  return result;
}

void handlePortal() {
  if (APserver.method() == HTTP_POST) {
    struct wifiSettings temp_user_wifi;
    strncpy(temp_user_wifi.ssid,     APserver.arg("ssid").c_str(),     sizeof(user_wifi.ssid));
    strncpy(temp_user_wifi.password, APserver.arg("password").c_str(), sizeof(user_wifi.password));
    temp_user_wifi.ssid[APserver.arg("ssid").length()] = temp_user_wifi.password[APserver.arg(
"password").length()] = '\0';
    if ((temp_user_wifi.ssid != user_wifi.ssid) || (temp_user_wifi.password != user_wifi.password))
 {
      user_wifi = temp_user_wifi;
      EEPROM.put(0, user_wifi);
      EEPROM.commit();
    }
    APserver.send(200, "text/html", "<!doctype html><html lang='en'><head><meta
charset='utf-8'><meta name='viewport' content='width=device-width, initial-scale=1'><title>Wifi
Setup</title><style>*,::after,::before{box-sizing:border-box;}body{margin:0;font-family:'Segoe
```

```
UI',Roboto,'Helvetica Neue',Arial,'Noto Sans','Liberation
Sans';font-size:1rem;font-weight:400;line-height:1.5;color:#212529;background-color:#f5f5f5;}.form-c
ontrol{display:block;width:100%;height:calc(1.5em + .75rem + 2px);border:1px solid
#ced4da;}button{border:1px solid
transparent;color:#fff;background-color:#007bff;border-color:#007bff;padding:.5rem
1rem;font-size:1.25rem;line-height:1.5;border-radius:.3rem;width:100%}.form-signin{width:100%;max-wi
dth:400px;padding:15px;margin:auto;}h1,p{text-align: center}</style></head><body><main
class='form-signin'><h1>Wifi Setup</h1><br><br><p>The wifi settings have been saved.<br><br>The
device will restarted.</p></main></body></html>" );
    delay(200);     // delay to allow the response to be send to the client..
    ESP.restart();  // ..before resetting for setup() to try the ssid, password
  } else {
    APserver.send(200, "text/html", "<!doctype html><html lang='en'><head><meta
charset='utf-8'><meta name='viewport' content='width=device-width, initial-scale=1'><title>Wifi
Setup</title> <style>*,::after,::before{box-sizing:border-box;}body{margin:0;font-family:'Segoe
UI',Roboto,'Helvetica Neue',Arial,'Noto Sans','Liberation
Sans';font-size:1rem;font-weight:400;line-height:1.5;color:#212529;background-color:#f5f5f5;}.form-c
ontrol{display:block;width:100%;height:calc(1.5em + .75rem + 2px);border:1px solid
#ced4da;}button{cursor: pointer;border:1px solid
transparent;color:#fff;background-color:#007bff;border-color:#007bff;padding:.5rem
1rem;font-size:1.25rem;line-height:1.5;border-radius:.3rem;width:100%}.form-signin{width:100%;max-wi
dth:400px;padding:15px;margin:auto;}h1{text-align: center}</style></head><body><main
class='form-signin'><form action='/' method='post'><h1 class=''>Wifi Setup</h1><br><div
class='form-floating'><label>SSID</label><input type='text' class='form-control'
name='ssid'></div><div class='form-floating'><br><label>Password</label><input type='password'
class='form-control' name='password'></div><br><br><button
type='submit'>Save</button></form></main> </body></html>" );
  }
}
void initOTA() {
// Port defaults to 8266, alternatively ArduinoOTA.setPort(your_port);
// Hostname defaults to esp8266-[ChipID], alternatively ArduinoOTA.setHostname("your_hostname");
// No authentication by default, alternatively ArduinoOTA.setPassword("admin");
// Password can be set with it's md5 value as well:
// MD5(admin) = 21232f297a57a5a743894a0e4a801fc3
// ArduinoOTA.setPasswordHash("21232f297a57a5a743894a0e4a801fc3");

    ArduinoOTA.onStart([]() {
            String type;
            if (ArduinoOTA.getCommand() == U_FLASH) {
                    type = "sketch";
            } else { // U_FS
                    type = "filesystem";
            }
    });

    ArduinoOTA.onEnd([]() {     });

    ArduinoOTA.onProgress([](unsigned int progress, unsigned int total) {
//          Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
    });

    ArduinoOTA.onError([](ota_error_t error) { //   Serial.printf("Error[%u]: ", error);
            if (error == OTA_AUTH_ERROR) { //       Serial.println("Auth Failed");
            } else if (error == OTA_BEGIN_ERROR) { //   Serial.println("Begin Failed");
            } else if (error == OTA_CONNECT_ERROR) { // Serial.println("Connect Failed");
            } else if (error == OTA_RECEIVE_ERROR) { // Serial.println("Receive Failed");
            } else if (error == OTA_END_ERROR) {// Serial.println("End Failed");
            }
    });
```

```
    ArduinoOTA.begin();
}
// ISR , handles rising edges from CF
void CF_impulse() { // Captures count of pulses from CF_PIN
  msNow = millis();
  pulseWidth = msNow - msPreviousPulse;
  msPreviousPulse = msNow;
  pulseFreq = float(1000 / pulseWidth);
}
```