Temps et Esp32

J. Lemaire

Octobre 2024

1 Les concepts liés au temps

1.1 Temps UTC

Le mot **temps** vient du latin **tempus** qui correspondait à une **durée**, c'est-à-dire le temps écoulé entre 2 événements. Mais cette définition qui « se mord la queue » montre que ce concept n'est pas facile à définir ! Le temps peut être considéré comme une « grandeur physique continue qui permet de situer la succession des événement dans un référentiel donné »[1].

Mesurer des temps a toujours représenté un challenge pour les scientifiques, notamment le temps moyen d'un **jour** solaire (entre 2 passages du soleil au zénith), chez les anciens Egyptiens qui comptaient en base 12 : ce sont eux qui ont introduit sa division en **heures** et constaté qu'elle restait pratiquement constante et égale à 2x12 heures. Idem pour la division d'une année (entre 2 maxima du soleil au zénith) en 12 **mois** d'une trentaine de jours environ. La division des heures en 60 **minutes** puis celle des minutes en 60 **secondes** seraient dues quant-à elles aux Babyloniens qui comptaient en base 60.

Ceci conduit « naturellement » à définir la **seconde** comme unité de temps égale à 1/86400 jour solaire moyen. Malheureusement ceci est imprécis car la durée d'un jour solaire, qui dépend de la rotation de la terre, varie en particulier sous l'influence des marées.

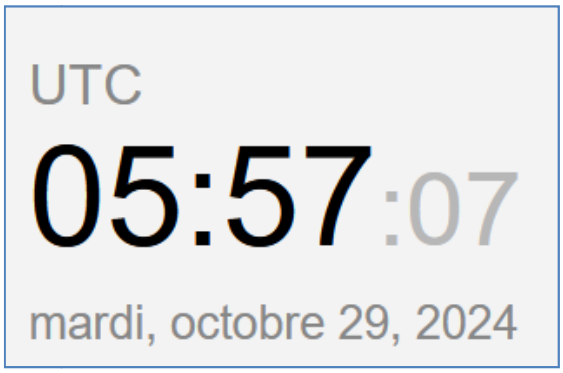
Les scientifique ont donc dû rechercher d'autres définitions, notamment celles basées sur des phénomènes quantiques : depuis 1967, la **seconde**, est définie comme étant égale à 9 192 631 770 fois la période de vibration (changement de niveau d'énergie) d'un atome de Césium 133, plongé dans un champ électromagnétique.

En synchronisant les horloges, il est donc possible de définir de manière uniforme la **date de l'instant présent**, c'est-à-dire le temps qui s'est écoulé depuis un événement de référence, historiquement la naissance de Christ, mais dans la pratique une date conventionnelle plus proche de nous. De nos jours, quasiment tous les pays ont adopté ainsi le **temps universel coordonné** (UTC)[2] compromis entre le **temps universel** (TU)[3], fondé sur la rotation de la terre, qui lui-même a remplacé le **temps moyen de Greenwich** (GMT)[4], basé sur l'observation du soleil au zénith et le **temps atomique international** (TAI)[5], basé sur plusieurs horloges au Césium 133 dans le monde. On notera au passage l'ambigüité de la terminologie, car il s'agit bien de **dates** et non de **temps**, mais on conservera cette terminologie conventionnelle.

---

[1] https://www.futura-sciences.com/sciences/definitions/physique-temps-325/
[2] https://fr.wikipedia.org/wiki/Temps_universel_coordonné
[3] https://fr.wikipedia.org/wiki/Temps_universel
[4] https://fr.wikipedia.org/wiki/Temps_moyen_de_Greenwich
[5] https://fr.wikipedia.org/wiki/Temps_atomique_international

La page Web https://24timezones.com/fuseau-horaire/utc fournit par exemple ce **temps UTC** :

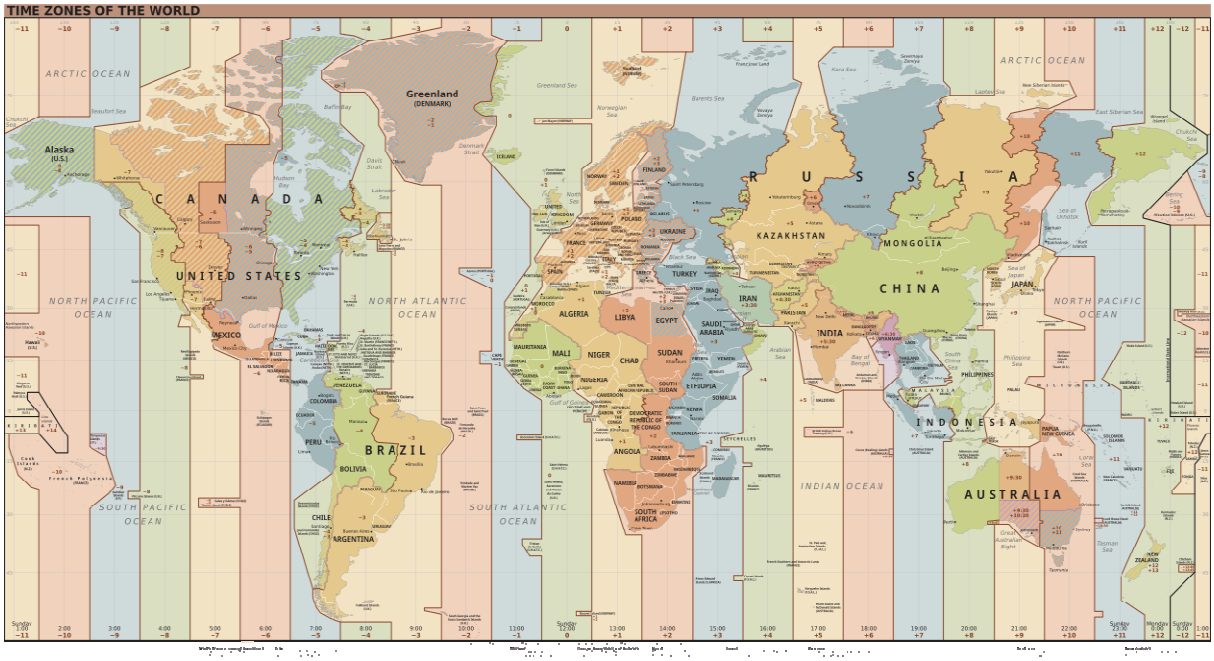


Si on veut l'obtenir dans une application informatique, on fera généralement appel à un **Serveur NTP**[6] sur Internet, en utilisant le protocole du même nom[7], sachant donc que tous ces serveurs délivrent la même information ; classiquement, on appelle ici **pool.ntp.org**[8] qui redirigera l'appel vers un serveur proche.

1.2 Temps LOCAL

Cette date uniforme de l'instant présent fournit une heure compatible avec le soleil (12h => zénith) sur le méridien de Greenwich, mais ce n'est évidemment pas le cas pour d'autres méridiens terrestres.

Les pays se sont donc mis d'accord pour définir 24 **fuseaux horaires** [9] correspondant approximativement à des **décalages permanents** de +/- 1h quand on passe d'un fuseau horaire au suivant ou au précédent :



[6] Network Time Protocol.
[7] https://fr.wikipedia.org/wiki/Network_Time_Protocol
[8] https://www.ntppool.org/zone/fr
[9] https://en.wikipedia.org/wiki/Time_zone

Mais, pour des raisons économiques, ils ont souvent ajouté des **décalages temporaires** de +1h pendant les mois de plus fort ensoleillement pour profiter de ce dernier : c'est le système des **heures d'été** ou **DST**[10].

Dans ce qui suit[11] :

décalage = **décalage permanent** + **décalage temporaire**

permet de définir :

temps LOCAL = **temps UTC** + **décalage**

Pour un lieu donné, ce **décalage** est précisé dans une chaîne de caractères, codée en format **POSIX**[12], dépendant de son pays et de son fuseau horaire. On pourra trouver toutes ces chaînes décrivant les **zones de temps**[13] sur le site :

https://github.com/nayarsystems/posix_tz_db/blob/master/zones.csv

Par exemple, celle de la **France métropolitaine** est décrite par la chaîne **Europe/Paris** :

"CET-1CEST,M3.5.0,M10.5.0/3"

Elle indique que le **décalage** à appliquer au **temps UTC** pour obtenir le **temps Local en France métropolitaine** sera de :

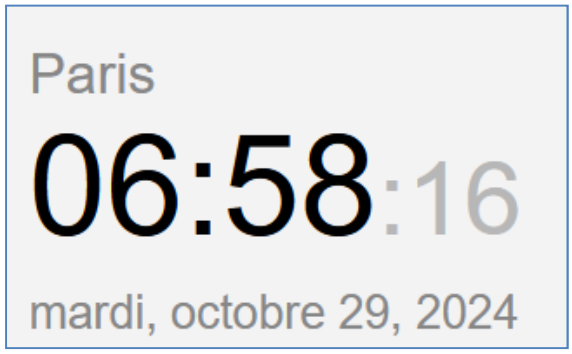
- +2h en heure d'été.
- +1h en heure d'hiver.

et on changera d'heure :

- Le dernier dimanche de mars à 2h du matin, en avançant d'1h toutes les pendules, pour passer à l'heure d'été.
- Le dernier dimanche d'octobre à 3h du matin, en retardant d'1h toutes les pendules, pour passer à l'heure d'hiver.

Ainsi, en 2024, les changements ont été effectués le 31 mars et le 27 octobre.

Là encore on peut utiliser le site précédent et sa page web https://24timezones.com/Paris/heure pour obtenir le **temps Local en France Métropolitain**e ; voici ce qu'il fournit, 1m9s après l'appel précédent :



[10] Daylight Saving Time.
[11] Offset UTC.
[12] https://docs.postgresql.fr/11/datetime-posix-timezone-specs.html
[13] **Time zone** en anglais ;

On notera qu'il y a bien un **décalage** d'**1h** entre le **temp UTC** et le **temps local**, puisque le 29/10/2024 est un jour en heures d'hiver dans cette **zone de temps**.

En France, ces **temps UTC** ou les **temps locaux** sont généralement affichés sous la forme :

JJ/MM/AAAA hh:mm:ss

Par exemple 29/10/2024 06:58:16 pour le **temps local** précédent. Mais ce n'est pas suffisant pour les temps locaux, lorsqu'on passe de l'heure d'été à l'heure d'hiver, comme on va le voir plus loin. C'est pour cette raison que la norme suivante a été introduite.

1.3 Norme ISO 8601

La norme **ISO 8601**[14] propose d'écrire les temps sous les formats suivants :

- **AAAA-MM-JJThh:mm:ssZ**[15] pour un **temps UTC**
- **AAAA-MM-JJThh:mm:sszzzzzz où zzzzzz=+/-hh:mm** précise le **décalage** pour un **temps local**

Par exemple, les 2 temps fournis précédemment s'écriront:

- **2024-10-29T05:57:07Z** pour le temps UTC
- **2024-10-29T07:58:16+01:00** pour le temps local

Cette indication du **décalage** (+1h ici) peut sembler superflue, mais elle ne l'est pas au moment du passage de l'heure d'été à l'heure d'hiver, car on aura la succession suivante de **temps locaux**:

- …
- 2024-10-27T02:00:00+02:00 (heure d'été)
- …
- 2024-10-27T02:59:59+02:00(heure d'été)
- 2024-10-27T02:00:00+01:00 (à 3h, on retarde d'une heure et on passe en heure d'hiver)
- …
- 2024-10-27T02:59:59+01:00 (heure d'hiver)
- …

Ainsi 27/10/2024 02h00m00s, …, 27/10/2024 02h59m59s désignent 2 temps différents selon qu'on se trouve en heure d'été ou heure d'hiver : il y a **ambiguïté** et il faudra donc ici indiquer le **décalage** pour préciser un temps dans cet intervalle de 2h.

Inversement, quand on passe de l'heure d'hiver à l'heure d'été, le 31/03/2024, on aura la succession suivante de temps locaux :

- …
- 2024-03-31T01:00:00+01:00(heure d'hiver)
- …
- 2024-03-31T01:59:59+01:00(heure d'hiver)
- 2024-03-31T03:00:00+02:00 (à 2h, on avance d'une heure et on passe en heure d'été)
- …

[14] https://fr.wikipedia.org/wiki/ISO_8601
[15] Z pour Zéro décalage.

- 2024-03-31T03:59:59+02:00 (heure d'été)
- …

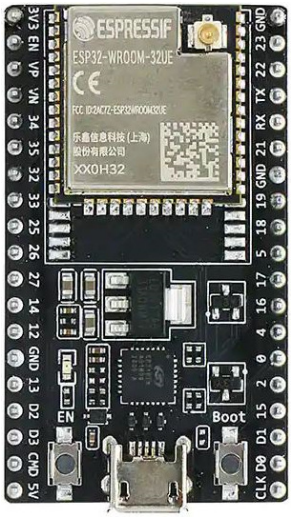
Cette fois, 31/03/2024 02h00m00s et 31/03/2024 03h00m00s, …, 31/03/2024 02h59m59s et 31/03/2024 03h59m59s désignent les mêmes temps, 2024-03-31T03:00:00+02:00, …, 2024-03-31T03:59:59+02:00, ce qui n'est pas trop gênant car ces derniers sont bien précisés[16].

Evidemment, hormis cette situation très particulière, on pourra généralement définir des **heures locales** sans ambiguïté, sans avoir à préciser le **décalage**.

2 Accès au temps avec un microcontrôleur

2.1 Carte ESP32 DevkitC V4[17]

Basée ici sur le module **Espressif ESP32-WROOM-32UE**, on peut y connecter une antenne externe 2,4GHz pour améliorer la qualité de la transmission Wifi :



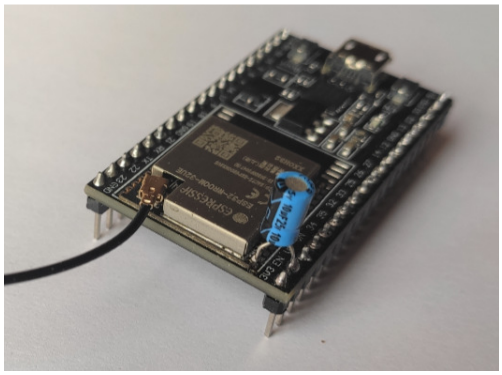
Elle a été programmée avec un PC sous Windows 10, en utilisant l'extension **PlatformIO** de **Visual Studio Code**[18].

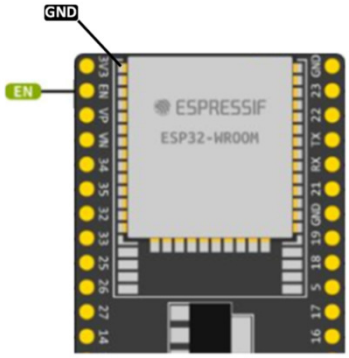
Pour y transférer un programme, il suffit de la connecter au PC via son port micro-USB, qui permet également de l'alimenter[19].

Par contre, ce transfert n'est possible que si la carte est un mode « bootloader » et il faut a priori maintenir enfoncé le bouton « Boot » pendant tout le transfert pour installer ce mode. On peut éviter cela en soudant un condensateur électrolytique de 10µF entre la broche EN de la carte et la patte GND de son module Espressif[20] :

---

[16] On pourrait également considérer comme inexisants les temps 31/03/2024 02h00m00s, …, 31/03/2024 02h59m59s.
[17] https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html#get-started-esp32-devkitc-board-front
[18] https://docs.platformio.org/en/stable/integration/ide/vscode.html
[19] Outre l'utilisation du port USB, on peut aussi l'alimenter en utilisant ses broches 5V et GND ou bien 3V3 et GND, mais à condition de ne choisir qu'une seule de ces 3 possibilités.
[20] https://randomnerdtutorials.com/solved-failed-to-connect-to-esp32-timed-out-waiting-for-packet-header/





2.2 Programme

2.2.1 Code

```
// Time - main.cpp

/*****************************************************************************

  Objective
    Time utilities for ESP32, that includes TZ and DST adjustments.
    Get the POSIX style TZ format string from  https://github.com/nayarsystems/posix_tz_db/blob/master/zones.csv

  References
  - Wifi :
    . https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/
  - Current time :
    . https://randomnerdtutorials.com/esp32-date-time-ntp-client-server-arduino/
    . https://randomnerdtutorials.com/esp32-ntp-timezones-daylight-saving/
    . https://sourceware.org/newlib/libc.html#Timefns
    . https://cplusplus.com/reference/ctime/
    . https://github.com/espressif/arduino-esp32/blob/master/cores/esp32/esp32-hal-time.c#L47
    . https://www.gnu.org/software/libc/manual/html_node/TZ-Variable.html

*****************************************************************************
  Libraries and types
*****************************************************************************/

#include <WiFi.h>
#include <time.h>

/*****************************************************************************
  Constants
*****************************************************************************/

// Local network access point
const char *SSID = "---------";
const char *PWD = "---------";

// NTP server (=>UTC time) and Time zone
const char* NTP_SERVER = "pool.ntp.org";  // Server address (or "ntp.obspm.fr", "ntp.unice.fr", ...)
const char* TIME_ZONE = "CET-1CEST,M3.5.0,M10.5.0/3"; // Europe/Paris time zone

/*****************************************************************************
  Tool functions
*****************************************************************************/

void setTimeZone(const char *timeZone)
{
  // To work with Local time (custom and RTC)
  setenv("TZ", timeZone, 1);
  tzset();
}

void initRTC(const char *timeZone)
{
  // Set RTC with Local time, using an NTP server
  configTime(0, 0, "pool.ntp.org"); // To get UTC time
  tm time;
  getLocalTime(&time);
  setTimeZone(timeZone);  // Transform to Local time
}

bool getCustomTime(int year, int month, int day, int hour, int minute, int second, tm *timePtr)
{
  // Set a time (date) without DST indication
  *timePtr = {0};
  timePtr->tm_year = year - 1900;
  timePtr->tm_mon = month-1;
```

```
  timePtr->tm_mday = day;
  timePtr->tm_hour = hour;
  timePtr->tm_min = minute;
  timePtr->tm_sec = second;
  time_t t = mktime(timePtr);
  timePtr->tm_hour--;
  time_t t1 = mktime(timePtr);
  memcpy(timePtr, localtime(&t), sizeof(tm));
  if (localtime(&t1)->tm_isdst==1)
  {
    if (timePtr->tm_isdst==0) return false;  // Ambigous
    else timePtr->tm_hour--;
  }
  return true;
}

void printTime(const char *str, tm *timePtr, bool ok)
{
  // Print time with DST indication (like ISO 8601)
  if (ok)
  {
    char buf[30];
    strftime(buf, sizeof(buf), "%Y-%m-%d %H:%M:%S %z", timePtr);
    Serial.printf("%s : %s\n", str, buf);
  }
  else Serial.printf("%s : is ambigous\n", str);
}

/*****************************************************************************
  setup and loop functions
*****************************************************************************/

void setup()
{
  // Open serial port
  Serial.begin(115200);
  while (!Serial);

  // Set time zone (not necessary after InitRTC)
  setTimeZone(TIME_ZONE);

  // Custom time
  struct tm time;
  bool okTime;
  Serial.println("\nCUSTOM TIME");
  Serial.println("Summer time ON");
  okTime = getCustomTime(2024, 3, 31, 1, 59, 59, &time); printTime("31/03/2024 01h59m59s", &time, okTime);
  okTime = getCustomTime(2024, 3, 31, 2, 0, 0, &time); printTime("31/03/2024 02h00m00s", &time, okTime);
  okTime = getCustomTime(2024, 3, 31, 2, 59, 59, &time); printTime("31/03/2024 02h59m59s", &time, okTime);
  okTime = getCustomTime(2024, 3, 31, 3, 0, 0, &time); printTime("31/03/2024 03h00m00s", &time, okTime);
  okTime = getCustomTime(2024, 3, 31, 3, 59, 59, &time); printTime("31/03/2024 03h59m59s", &time, okTime);
  okTime = getCustomTime(2024, 3, 31, 4, 0, 0, &time); printTime("31/03/2024 04h00m00s", &time, okTime);
  Serial.println("Summer time OFF");
  okTime = getCustomTime(2024, 10, 27, 1, 59, 59, &time); printTime("27/10/2024 01h59m59s", &time, okTime);
  okTime = getCustomTime(2024, 10, 27, 2, 0, 0, &time); printTime("27/10/2024 02h00m00s", &time, okTime);
  okTime = getCustomTime(2024, 10, 27, 2, 59, 59, &time); printTime("27/10/2024 02h59m59s", &time, okTime);
  okTime = getCustomTime(2024, 10, 27, 3, 0, 0, &time); printTime("27/10/2024 03h00m00s", &time, okTime);
  Serial.println("Automatic correction");
  okTime = getCustomTime(2023, 2, 29, 0, 0, 0, &time); printTime("29/02/2023 00h00m00s", &time, okTime);
  Serial.println("\nLOCAL TIME");

  // Connect to the Wifi access point
  WiFi.begin(SSID, PWD);
  while (WiFi.status() != WL_CONNECTED);
  Serial.printf("IP=%s RSSI=%d\n", WiFi.localIP().toString(), WiFi.RSSI());

  // Init RTC with Local time using an NTP server
  initRTC(TIME_ZONE);
  WiFi.disconnect(true);

  // Local time from RTC (without Wifi)
        getLocalTime(&time); printTime("now", &time, true);
  delay(1000);
  getLocalTime(&time); printTime("now+1s", &time, true);
}

void loop()
{
}
```

2.2.2 Résultats affichés

```
CUSTOM TIME
Summer time ON
31/03/2024 01h59m59s : 2024-03-31 01:59:59 +0100
31/03/2024 02h00m00s : 2024-03-31 03:00:00 +0200
31/03/2024 02h59m59s : 2024-03-31 03:59:59 +0200
```

```
31/03/2024 03h00m00s : 2024-03-31 03:00:00 +0200
31/03/2024 03h59m59s : 2024-03-31 03:59:59 +0200
31/03/2024 04h00m00s : 2024-03-31 04:00:00 +0200
Summer time OFF
27/10/2024 01h59m59s : 2024-10-27 01:59:59 +0200
27/10/2024 02h00m00s : is ambigous
27/10/2024 02h59m59s : is ambigous
27/10/2024 03h00m00s : 2024-10-27 03:00:00 +0100
Automatic correction
29/02/2023 00h00m00s : 2023-03-01 00:00:00 +0100

LOCAL TIME
IP=192.168.1.19 RSSI=-83
now : 2024-10-28 18:49:14 +0100
now+1s : 2024-10-28 18:49:15 +0100
```

2.2.3 Explications

- Pour manipuler des temps sur ESP32, en C++, on utilise généralement la bibliothèque **Time**, issue du langage C. Elle est intégrée au système de base et permet de manipuler les temps, en utilisant des instances de la structure **tm** ; ses principaux champs sont :

Membre	Description
tm_sec	Secondes
tm_min	Minutes
tm_hour	Heures
tm_mday	Jour du mois
tm_mon	Mois - 1
tm_year	Année - 1900
tm_isdst	1 si heure d'été, 0 sinon

  On considèrera également des instances de **time_t**, qui représenteront le nombre de secondes depuis le 1er janvier 1970 à 00h00m00s, les fonctions de conversion étant :

  - `time_t mktime (struct tm *_timeptr);`
  - `struct tm *localtime (const time_t *_timer);`

- Pour afficher des temps, on utilise souvent la fonction **strftime** ; par exemple, les instructions

```
char buf[30];
strftime(buf, sizeof(buf), "%Y-%m-%d %H:%M:%S %z", timePtr);
Serial.printf("%s : %s\n", str, buf);
```

  produisent un affichage presque conforme à la **norme ISO 8601**, lorsque **timePtr** contient l'adresse d'une instance de **tm** ; par exemple :

```
2024-01-01 00:00:00 +0100
```

  C'est ce qu'on fait dans la fonction **printTime**.

- Le gros intérêt des fonctions **mktime** et **localtime** est qu'elles peuvent corriger des erreurs d'une instance de la classe **tm**, en les enchaînant, et ceci en tenant compte du **décalage** horaire d'une **zone de temps** : c'est exactement ce qu'on met à profit dans la fonction suivante :

```
bool getCustomTime(int year, int month, int day, int hour, int minute, int second, tm
*timePtr)
```

Elle écrit à l'adresse **timePtr** les champs corrects et en particulier **tm_isdst**[21] ; par exemple:

```
getCustomTime(2024, 2, 30, 0, 0, 0, &time) => 2024-03-01 00:00:00 +0100
```

car en France métropolitaine, en 2024 qui est une année bissextile, février n'a que 29 jours.

- La bibliothèque **Time** offre également un moyen très simple pour préciser les paramètres d'une **zone de temps** (**décalage** + dates de changement de ce dernier). C'est ce qui est mis en œuvre dans la fonction suivante :

```
void setTimeZone(const char *timeZone)
{
  // To work with Local time (custom and RTC)
  setenv("TZ", timeZone, 1);
  tzset();
}
```

Dans notre cas, on lui passera la chaîne **Europe/Paris** : "CET-1CEST,M3.5.0,M10.5.0/3".

- On notera bien que tout ceci n'utilise pas l'**horloge RTC** interne de l'ESP32. Par contre, si on veut connaître l'heure locale, à tout moment, il suffit de mettre à profit cette fonctionnalité des ESP32, en l'initialisant avec le **temps UTC** fourni par un serveur NTP sur Internet, puis là encore de faire appel à **setTimeZone** pour le transformer en **temps local**. C'est ce qui est fait dans la fonction suivante :

```
void initRTC(const char *timeZone)
{
  // Set RTC with Local time, using an NTP server
  configTime(0, 0, "pool.ntp.org"); // To get UTC time
  tm time;
  getLocalTime(&time);
  setTimeZone(timeZone);  // Transform to Local time
}
```

L'accès à Internet se fera en **Wifi**[22], autre fonctionnalité de base sur un ESP32. Mais on notera bien qu'ensuite, la Wifi n'est plus utile et qu'il suffit d'utiliser à nouveau l'instruction :

```
getLocalTime(&time);
```

pour obtenir le **temps local** mis à jour par l'**horloge RTC** interne, qui fonctionnera tant que l'ESP32 sera actif. Et là encore, on profitera de toute la fonctionnalité de gestion du décalage horaire.

[21] Sauf pour une plage horaire de 2h, quand on passe à l'heure d'hiver (cf. §1.3).
[22] Il faudra pour cela renseigner les constantes **SSID** et **PWD** dans le programme.