Préalable : IL FAUT QUE TOUT LE MONDE SOIT CONNECTEE AU MEME RESEAU

Table des matières

[Explication du fonctionnement de l'OTA sur un ESP32 (Pour Moi en soi, une sorte de rappel) 2](#_Toc189087656)

[Code Exemple 3](#_Toc189087657)

[Étapes pour configurer le terminal sur VSCode 4](#_Toc189087658)

[Activer Telnet sur Windows 5](#_Toc189087659)

[Annexe 6](#_Toc189087660)

# Explication du fonctionnement de l'OTA sur un ESP32 (Pour Moi en soi, une sorte de rappel)

**Introduction à la programmation OTA (Over-the-Air)**

La programmation OTA permet de mettre à jour le code des cartes ESP32 sans avoir besoin de les connecter physiquement à un ordinateur via un port série. En utilisant l'OTA , il est possible de téléverser du nouveau code sur l'ESP32 directement depuis un ordinateur qui sont connectés au même réseau local.

**Principe de fonctionnement de l'OTA**

1. **Premier téléversement via port série** : Le premier programme doit être téléversé sur l'ESP32 via une connexion série traditionnelle. Ce programme contient le code qui permet d'activer la fonctionnalité de mise à jour OTA.
2. **Téléversement via OTA** : Pour chaque mise à jour ultérieure, il est nécessaire d'intégrer le code OTA dans tous les nouveaux programmes pour permettre celui-ci d’être toujours présent dans la mémoire flash. Cela garantit que vous pourrez continuer à téléverser du code via OTA à l'avenir.
3. **Limite** : Si vous téléversez un code qui ne contient pas la routine OTA, vous perdrez la possibilité de mettre à jour l'ESP32 via OTA. Il faudra alors recommencer en utilisant la connexion série.

**OTA dans le Bootloader de l'ESP32**

Le processus de mise à jour OTA sur l'ESP32 repose sur le bootloader, qui est le premier logiciel exécuté au démarrage. Ce bootloader gère les partitions de mémoire nécessaires pour effectuer des mises à jour OTA de manière fiable et sécurisée.

**Configuration des Partitions**

Pour le bon fonctionnement de l'OTA, l'ESP32 dispose de plusieurs partitions dans sa mémoire flash :

* **Partitions d'application** : Deux partitions de firmware sont généralement présentes, souvent appelées ota\_0 et ota\_1. L'une est active et exécute le firmware actuel, tandis que l'autre est prête à recevoir de nouvelles mises à jour.
* **Partition de Données OTA** : Une partition spéciale, appelée *OTA data*, stocke des informations sur la dernière mise à jour réussie. Le bootloader lit cette partition au démarrage pour déterminer quelle image de firmware charger. Si aucune donnée valide n'est trouvée, l'image d'usine est chargée.

**Processus de Mise à Jour OTA**

1. **Démarrer le processus OTA** : L'OTA commence par effacer la partition cible pour faire de la place pour le nouveau firmware.
2. **Écriture du nouveau firmware** : Le nouveau firmware est écrit dans la partition inactive, assurant que le fichier est correctement transféré.
3. **Terminer la mise à jour** : Une fois l'écriture terminée, l'intégrité de l'image nouvellement écrite est validée pour garantir son exactitude.
4. **Changer la partition de démarrage** : La partition de données OTA est mise à jour pour pointer vers la nouvelle partition contenant le firmware mis à jour.

**Redémarrage et Vérification**

Après la mise à jour OTA, le microcontrôleur redémarre. Le bootloader effectue une vérification de l'intégrité du nouveau firmware.

* Si la mise à jour est réussie et valide, le bootloader bascule vers la nouvelle partition, chargeant ainsi le nouveau firmware lors du prochain démarrage.
* En cas d'échec de la mise à jour (par exemple, si le fichier est corrompu), le bootloader retourne à la partition précédente, assurant que le microcontrôleur ne soit pas "briqué".

# Code Exemple

Voir annexe.

Dans le fichier platform ini, il faudra spécifier cette commande-ci, cela signifie permet à platform io de savoir à qui envoyé le programme :

upload\_port = BaseEsp32.local

Également d’installer la bibliothèque permettant d’effectuer une « liaison série » mais via le wifi avec le protocole telnet :

lib\_deps =

[jandrassy/TelnetStream@^1.3.0](mailto:jandrassy/TelnetStream@%5e1.3.0)

Ainsi que de bien vérifier que le baudrate paramétrer sois le même que celui dans le fichier de configuration platform ini : monitor\_speed = 115200

# Étapes pour configurer le terminal sur VSCode

1. **Installer les Extensions Nécessaires** :
   * Assurez-vous d'avoir installé l'extension **PlatformIO** ou **Arduino** pour VSCode. Ces extensions facilitent la gestion des projets et l'utilisation du moniteur série.
2. **Ouvrir un terminal sur vs code ou d’une autre manière** :

Pour établir une connexion Telnet avec votre ESP32 depuis le terminal de VSCode, vous pouvez procéder comme suit :

* + Dans NOTRE cas, l'ESP32 est accessible via le wifi depuis le protocole telnet, donc vous devez vous connecter à l'adresse BaseEsp32.local (car c’est celle-ci que j’ai spécifier dans le code et dans platform io) sur le port 23.
  + Allez dans Affichage > Terminal (ou appuyez sur Ctrl+`` `) pour ouvrir le terminal intégré de VSCode.
  + Dans le terminal, tapez la commande suivante pour vous connecter à votre ESP32 : telnet BaseEsp32.local 23**.** Sur le code exemple, si on est connecté en liaison série, la commande apparait d’elle-même pour guider l’utilisateur

**Configuration du Moniteur Série**

Pour les paramètres de communication, vous pouvez définir les paramètres suivants :

* **Vitesse de transmission (Baud rate)** **(Cas en liaison Série physique)** : Cela dépend de votre configuration dans le code de l'ESP32. Un réglage commun est 115200, mais vérifiez votre code pour confirmer.
* **Port(Cas en liaison Wifi par exemple)** : Comme mentionné, l'adresse sera BaseEsp32.local et le port 23 pour Telnet

# Activer Telnet sur Windows

1. **Ouvrir le Panneau de Configuration** :
   * Cliquez sur le bouton **Démarrer** et tapez "Panneau de configuration", puis appuyez sur **Entrée**.
2. **Accéder aux Programmes et Fonctionnalités** :
   * Cliquez sur **Programmes** puis sur **Activer ou désactiver des fonctionnalités Windows**.
3. **Activer le Client Telnet** :
   * Dans la fenêtre qui s'ouvre, faites défiler la liste jusqu'à trouver **Client Telnet**.
   * Cochez la case à côté de **Client Telnet**.
   * Cliquez sur **OK** pour installer le client.
4. **Redémarrer le Terminal** :
   * Après l’installation, fermez et rouvrez le terminal dans Visual Studio Code ou utilisez une nouvelle fenêtre de terminal.

Voici un lien vers une vidéo qui montre comment activer Telnet sur Windows. <https://www.youtube.com/watch?v=Q2sNkTJv2fc>

**Tester la Connexion Telnet**

Une fois Telnet activé, vous devriez pouvoir exécuter la commande suivante dans le terminal : telnet BaseEsp32.local 23

Dans mon cas, j'avais nommé mon ESP32 *BaseEsp32* pour l'OTA et le Wi-Fi.

Le 23 est le port choisi.  
  
Veuillez trouver ci-joint une vidéo (nommé Mise en place du monitoring dans le terminal) montrant la démarche pour téléverser le programme et accéder au terminal, comme si vous étiez en liaison série.

À la fin de la vidéo, mes gestes ne sont pas visibles, mais j'appuie sur la touche flèche haut jusqu'à retrouver ma commande *telnet*, puis je clique sur Entrée pour exécuter le programme.

La vidéo dite ci-joint est celle-ci : <https://youtu.be/c9qSqf4Abpc>

# Annexe

#include <Arduino.h>

#include <WiFi.h>

#include <ESPmDNS.h>

#include <WiFiUdp.h>

#include <ArduinoOTA.h>

#include <TelnetStream.h>

// Je laisse le routeur choisir l'adresse IP et

// je fixe le nom de l'adresse local de l'esp32 sur mon réseau Wi-Fi, ce qui simplifie les démarches, notamment lors de l'upload du code

//  Informations de connexion WiFi

const char \*name\_card\_elec = "BaseEsp32";

// BESOIN DE ME SIMPLIFIER MA VIE

#ifdef MON\_TELEPHONE                // Nom d'hôte de la carte ESP32

const char \*ssid = "Me voici";      // SSID du réseau WiFi

const char \*password = "youssef13"; // Mot de passe du réseau WiFi

#endif

#ifdef MA\_FREEBOX                                                  // Nom d'hôte de la carte ESP32

const char \*ssid = "Freebox-587F87";                               // SSID du réseau WiFi

const char \*password = "subcrescat-degend@-parciore@2-adducturos"; // Mot de passe du réseau WiFi

#endif

#ifdef Mon\_PC                                                      // Nom d'hôte de la carte ESP32

const char \*ssid = "Freebox-587F87";                               // SSID du réseau WiFi

const char \*password = "subcrescat-degend@-parciore@2-adducturos"; // Mot de passe du réseau WiFi

#endif

// Fonction pour gérer les opérations OTA dans une tâche séparée

void ota\_handle(void \*parameter)

{

  TickType\_t xLastWakeTime;

  xLastWakeTime = xTaskGetTickCount();

  for (;;)

  {

    receptionWIFI();

    // Gestion des opérations OTA (vérifie si une mise à jour est en cours)

    ArduinoOTA.handle();

    vTaskDelayUntil(&xLastWakeTime, pdMS\_TO\_TICKS(3500));

  }

}

// Fonction pour configurer l'OTA

void setupOTA()

{

  // Définit le nom d'hôte pour la carte ESP32 sur le réseau

  WiFi.setHostname(name\_card\_elec);

  // Passe le WiFi en mode station (client du réseau WiFi)

  WiFi.mode(WIFI\_STA);

  // Démarre la connexion WiFi avec les identifiants donnés

  WiFi.begin(ssid, password);

  Serial.println();

  Serial.print("SSID : ");

  Serial.println(ssid);

  Serial.println("Connexion au WiFi en cours...");

  // Boucle jusqu'à ce que la connexion au WiFi soit réussie

  while (WiFi.waitForConnectResult() != WL\_CONNECTED)

  {

    Serial.println("Connection Failed! Rebooting...");

    delay(1000);

    // Redémarrage si la connexion échoue (commenté pour l'instant)

    // ESP.restart();

  }

  // Affiche le succès de la connexion

  Serial.println("");

  Serial.println("Connexion établie !");

  // Définition du port OTA (par défaut, il est à 3232)

  // ArduinoOTA.setPort(3232);

  // Définit le nom d'hôte pour les opérations OTA

  ArduinoOTA.setHostname(name\_card\_elec);

  // Pas de mot de passe par défaut pour l'OTA

  // Configuration des différents événements de l'OTA

  ArduinoOTA

      .onStart([]() // Quand une mise à jour démarre

               {

      String type;

      // Si le type de mise à jour est pour le sketch (programme) ou SPIFFS (système de fichiers)

      if (ArduinoOTA.getCommand() == U\_FLASH)

        type = "sketch";

      else // U\_SPIFFS

        type = "filesystem";

      // Si on met à jour SPIFFS

      Serial.println("Start updating " + type); })

      .onEnd([]() // Quand la mise à jour est terminée

             { Serial.println("\nEnd"); })

      .onProgress([](unsigned int progress, unsigned int total) // Affiche le progrès de la mise à jour

                  { Serial.printf("Progress: %u%%\r", (progress / (total / 100))); })

      .onError([](ota\_error\_t error) // Gestion des erreurs lors de la mise à jour OTA

               {

      Serial.printf("Error[%u]: ", error);

      // Affiche un message d'erreur spécifique en fonction du type d'erreur

      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"); });

  // Démarre le service OTA

  ArduinoOTA.begin();

  // Démarre la communication série WiFi

  TelnetStream.begin();

  // Affiche les informations une fois prêtes

  Serial.println("Ready");

  Serial.print("IP address: ");

  Serial.println(WiFi.localIP()); // Affiche l'adresse IP attribuée

  Serial.print("Wifi Hostname: ");

  Serial.println(WiFi.getHostname()); // Affiche le nom d'hôte du WiFi

  Serial.print("Arduino Hostname: ");

  Serial.print(ArduinoOTA.getHostname()); // Affiche le nom d'hôte pour OTA

  Serial.println(".local");

  Serial.println("Commande a tapé dans le terminal");

  Serial.print("telnet ");

  Serial.print(ArduinoOTA.getHostname());

  Serial.println(".local 23");

  // Création d'une tâche FreeRTOS pour gérer l'OTA dans un thread séparé

  xTaskCreate(

      ota\_handle,   /\* Fonction de la tâche à exécuter \*/

      "OTA\_HANDLE", /\* Nom de la tâche \*/

      10000,        /\* Taille de la pile en octets \*/

      NULL,         /\* Paramètre passé à la tâche (aucun ici) \*/

      1,            /\* Priorité de la tâche \*/

      NULL);        /\* Handle de la tâche (aucun ici) \*/

}

void receptionWIFI()

{

  if (TelnetStream.available())

  {

    String input = TelnetStream.readString(); // Lire la chaîne complète

    Serial.print("Reçu: ");

    Serial.println(input); // Afficher ce qui a été reçu

    // Supprimer les espaces ou caractères indésirables (comme le retour à la ligne)

    input.trim(); // Enlève les espaces superflus et les retours à la ligne

    if (input.equals("STOP")) // Vérifier si l'entrée est "stop"

    {

      TelnetStream.println("STOP BASE ROULANTE");

    }

    if (input.equals("start")) // Vérifier si l'entrée est "start"

    {

      TelnetStream.println(i++); // Incrémenter i et envoyer

      Serial.println("Incrémenté : " + String(i));

    }

  }

}

// Fonction d'initialisation, exécutée une seule fois au démarrage

void setup()

{

    // Initialisation de la communication série à 115200 bauds

    Serial.begin(115200);

    Serial.println("Booting with OTA"); // Message indiquant le démarrage avec OTA

    // Appel à la fonction de configuration OTA (non définie dans ce code, mais probablement ailleurs)

    setupOTA();

    // Boucle jusqu'à ce qu'un client soit connecté via le port série WiFi

    while (!TelnetStream.available())

    {

        delay(500);                                             // Attente de 500 ms avant de vérifier à nouveau

        Serial.println("Aucun client connecté, en attente..."); // Message indiquant qu'il n'y a pas de client connecté

    }

    TelnetStream.println(""); // Pour qu'on puisse nous, de notré coté voir ce qu'on tape dans l'invite de commandes

    // Une fois qu'un client est connecté, envoie "Coucou" via la connexion série WiFi

    TelnetStream.println("Coucou");

}

// Boucle principale, exécutée en continu après le setup

void loop()

{

}